CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Integration of micro- and macroscopic models for pedestrian evacuation simulation
Authors	Murphy, Seán Óg
Publication date	2014
Original Citation	Murphy, S. O. 2014. Integration of micro- and macroscopic models for pedestrian evacuation simulation. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2014, Seán Óg Murphy. - http://creativecommons.org/licenses/by-nc-nd/3.0/
Download date	2026-11-16 04:23:08
Item downloaded from	https://hdl.handle.net/10468/1916



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Integration of Micro- and Macroscopic Models for Pedestrian Evacuation Simulation

Seán Óg Murphy
Dept. of Computer Science,
National University of Ireland, Cork

January 2014



Thesis Submitted to the National University of Ireland in Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Science.

Research Supervisors: Dr. Kenneth N. Brown, Prof. Cormac J. Sreenan
Head of Department: Prof. Barry O'Sullivan

Contents

1 Introduction **1**

1.1 Planning and Simulation of Emergency Evacuation 1

1.1.1 Thesis statement 1

1.2 Integration of Micro- and Macroscopic models 2

1.2.1 Macroscopic Pedestrian Modelling 3

1.2.2 Microscopic Pedestrian Modelling 3

1.3 Integration 4

1.4 Outline 5

2 Related Work **7**

2.1 Introduction 7

2.2 Sensing in Smart Buildings 8

2.3 Building models 11

2.3.1 Feature Detection 12

2.4 Macroscopic Models 13

2.4.1 Macroscopic Emergency Simulation 13

2.4.2 Graphs and Path-Planning 14

2.4.3 Dynamic Evacuation Planning and Guidance 16

2.5 Microscopic Pedestrian Simulation 18

2.5.1 Microscopic Agent Models 19

2.5.2 Agent behaviour and cognition 22

2.6 Hierarchical Microscopic and Macroscopic Models 24

2.7 Pedestrian Movement Data 26

2.8 Hazard modelling 28

2.9 Conclusions 29

3 Details and Validation of the Microscopic Agent Movement Model **30**

3.1 Introduction 30

3.2 Requirements 30

3.3 Model Details 31

3.3.1 Agent Tick procedure 33

3.3.2 Maintaining Personal Space 33

3.3.3 Steering towards Goals 34

3.4 Avoiding Obstacles 34

3.4.1 Braking and Avoiding collisions between Agents 35

3.4.2 Resolving Overlap ("Enforce core") 36

3.5 Validation Experiments (Bottleneck) 38

3.6 Validation Experiments (T-junction) 39

3.6.1 T-junction experiments with default agent 39

3.7 T-junction experiments with stream formation behaviour 41

3.7.1 T-junction Heatmaps . . . 45
3.8 Conclusions . . . 46

4 Hierarchical Level-of-Detail graphs for Evacuation Simulation and Planning 48
4.1 Introduction . . . 48
4.2 Graph Detail Levels and Requirements . . . 48
4.2.1 Microscopic Model Graph Requirements . . . 48
4.2.2 Macroscopic Planner Graph Requirements . . . 49
4.2.3 Hierarchical Detail levels . . . 50
4.3 Graph Generation Overview . . . 50
4.4 Fundamental graph generation . . . 51
4.4.1 Fundamental Graph . . . 52
4.5 Graph Simplification (Planner Graph) . . . 55
4.5.1 Loop Removal . . . 55
4.5.2 Triangle Removal . . . 58
4.5.3 Planner Graph Simplification Results . . . 59
4.6 Increasing Detail (Movement Graph) . . . 62
4.7 Structural graph simplification . . . 66
4.8 Graph Complexity (A* Running Time) . . . 70
4.9 Hybrid Path Planning for Agents . . . 70
4.10 Conclusions . . . 72

5 Micro/Macro Coupling of Flow Characteristics in Network Flow Graphs for Prediction and Planning 74
5.1 Introduction . . . 74
5.2 Network Flow Graph parameters . . . 74
5.3 Flow Coupling . . . 75
5.4 Path Testing in buildings . . . 78
5.5 Traversal Time Prediction . . . 82
5.6 Flow Graph Limitations . . . 84
5.6.1 Crowd Merging . . . 85
5.6.2 Junction Contraflow . . . 86
5.6.3 Direct Contraflow . . . 87
5.6.4 Crossroad Junction . . . 88
5.6.5 Possible Approaches for Overcoming Flow Graph Limitations 90
5.7 Simulated Evacuation using a Flow-Based Planner and coupled graphs 91
5.7.1 Introduction . . . 91
5.8 EvacPlan Dynamic Evacuation Planner . . . 92
5.8.1 Node Occupancy . . . 92
5.8.2 Edge Occupancy . . . 93

5.8.3 Hazard Locations . . . 93
5.8.4 Exit Locations . . . 94
5.8.5 Experiment setup . . . 94
5.8.6 Plan Allocation Strategies and Building Setup . . . 95
5.9 EvacPlan Experiment Results . . . 96
5.9.1 Injury count . . . 98
5.9.2 Evacuation performance over time . . . 100
5.10 Conclusions . . . 101

6 Exploiting Macroscopic Models for Microscopic Simulation and Sensing of Multiple Future States 105
6.1 Introduction . . . 105
6.2 Problem Decomposition . . . 106
6.2.1 Overview and Assumptions . . . 107
6.2.2 Route Allocation . . . 108
6.2.3 Complexity and orthogonality . . . 109
6.2.4 Discovering Orthogonal Future States . . . 110
6.2.5 Instance generation . . . 112
6.2.6 Decomposition Experiments . . . 113
6.2.7 Decomposition Results . . . 114
6.3 Simulation for Event Prediction and Sensing . . . 116
6.4 Candidate Event Simulation . . . 117
6.4.1 Assumptions . . . 117
6.4.2 Candidate Event Confidence Calculation . . . 118
6.4.3 Pattern Matching Experiments . . . 120
6.4.4 Pattern Matching Results . . . 121
6.5 Conclusions . . . 123

7 Conclusions and Future Work 125
7.1 Conclusions . . . 125
7.2 Future Work . . . 126

A EvacSim Pedestrian Evacuation Simulator 129
A.1 Introduction . . . 129
A.2 Simulation World Model . . . 129
A.2.1 Coordinate System . . . 130
A.2.2 Walls . . . 131
A.2.3 Line-of-Sight and Collisions . . . 131
A.2.4 Room Detection . . . 132
A.2.5 Traversability Graph . . . 135
A.2.6 Building Definition Files . . . 139
A.2.7 Building Drawing Tool . . . 141

A.3 Dynamic Simulation Elements . . . 142
A.3.1 Simulation Time and Ticks . . . 142
A.3.2 Occupant Agents . . . 144
A.3.3 Collision Correction . . . 146
A.3.4 Agent Path Planning . . . 147
A.3.5 Look-ahead and Re-planning . . . 147
A.3.6 Sensors and Network Nodes . . . 147
A.3.7 Network Node and Network . . . 147
A.3.8 Extensions to Network Node . . . 148
A.4 Scalability and Metrics . . . 149
A.4.1 Layers . . . 149
A.4.2 Agent Recording . . . 149
A.4.3 Agent Playback . . . 150
A.4.4 Geospatial Indexing . . . 150
A.4.5 Runtime performance . . . 151
A.4.6 Simulation Metrics . . . 152
A.4.7 Real-time Heatmaps . . . 153
A.4.8 High-detail Heatmaps using Voronoi Cells . . . 154

List of Tables

1 EvacSim agent variables 32
2 Graph Simplification results for irregularly shaped building 69
3 Group Traversal time estimate accuracy, first arrivals 83
4 Group Traversal time estimate accuracy, last arrivals 84
5 Crowd Merging traversal time prediction results 86
6 Junction Contraflow traversal time prediction results 87
7 Direct Contraflow traversal time prediction results 89
8 Crossroad Junction traversal time prediction results 90
9 Average Evacuation Time for evacuation experiments 98
10 Average number of injured agents at time of 95% evacuation 99
11 CrowdRoute merging example 113
12 Problem decomposition experiment results 115

List of Figures

1 Binary Space Partition and BSP Tree 12
2 Flocking agents forming an α-lattice 20
3 Zhang [1] T-junction real-world pedestrian experiment 27
4 Agent Lookahead for Avoiding Obstacles 35
5 Agent braking to avoid collision 36
6 Dense crowds without core enforcement. 37
7 Enforcing Agent core in dense crowds 37
8 Throughput Achieved for 2m corridor 38
9 Bottleneck throughput comparison of EvacSim with real-world experiment results 39
10 Zhang (left) and EvacSim (right) T-junction experiments 40
11 Flow vs Density in real-world T-junction experiments (taken from Zhang [2]) 41
12 EvacSim T-junction Flow vs Density (default agent model) 42
13 T-junction Flow vs Density (Stream-forming agent model) 43
14 Zhang [1] and EvacSim agent t-junction Flow vs Density comparison 44
15 Average Velocity Heatmaps for EvacSim Stream-forming agents, MDA=20, 60 45
16 Average Density Heatmaps for EvacSim Stream-forming agents, MDA=20, 60 45
17 Density and Velocity heatmaps for Zhang [1] T-junction experiments 46
18 Agent skipping ahead on a route (EvacSim screenshot) 49
19 Fundamental Graph and Rooms (B1) 53
20 Fundamental Graph and Rooms (B2) 54
21 Fundamental Graph and Rooms (Nimbus) 55
22 Graph Complexity (node count), fundamental graph 56
23 Graph Complexity (edge count), fundamental graph 56
24 Path Length Accuracy for Fundamental Graph 57
25 Loop Removal Graph Simplification 57
26 Triangle Removal Graph Simplification 58
27 Planner Graphs (Loop and Triangle Removal) 60
28 Planner Graph Complexity (node count) 61
29 Planner Graph Complexity (edge count) 61
30 EvacPlan running time using Planner Graphs 62
31 Path Length Accuracy, with Planner Graphs 62
32 Adding Corner Nodes on convex corners 63
33 Movement Graphs (Convex Corner Nodes) 65
34 Movement Graph Node Complexity 66
35 Movement Graph Edge Complexity 66

36 Path Length Accuracy (Movement Graph) . . . 67
37 Structural Graphs (merging nodes behind Gateways) . . . 68
38 Structural Graph Complexity . . . 69
39 Graph simplification on an irregularly shaped building . . . 70
40 Summary of A* Planning Computation Time results . . . 71
41 Path Length Accuracy (Hybrid Planning using plans provisioned by dynamic evacuation planner) . . . 72
42 Path Length Accuracy (Coupling Edges) . . . 77
43 EvacSim screenshots of two examples of path testing in progress . . . 78
44 Path testing: Capacity estimate accuracy . . . 79
45 Path testing: Capacity Real Difference . . . 79
46 Path angle changes and error (geometry occlusion) . . . 80
47 Error and Turning (changle in angle) . . . 80
48 Path Testing: Traversal Time Accuracy . . . 81
49 Path testing: Traversal Time Real Difference . . . 82
50 Group Traversal Time Experiment screenshot . . . 83
51 Simultaneous Flow through junction experiment scenarios . . . 85
52 EvacSim Screenshots: Crowd Merge Experiment (start, middle, end) . 85
53 Crowd Merge Experiment (Velocity Heatmap, EvacSim screenshot) . 86
54 EvacSim Screenshots: Junction Contraflow Experiment (start, middle, end) . . . 87
55 Junction Contraflow Experiment (Velocity Heatmap) . . . 87
56 EvacSim Screenshots: Direct Contraflow Experiment (start, middle, end) . . . 88
57 Direct Contraflow Experiment (Velocity Heatmap) . . . 88
58 EvacSim Screenshots: Crossroad Experiment (start, early, late, end) . 89
59 Crossroad Experiment (Velocity Heatmap) . . . 90
60 Loop Edge for detecting interaction of flows at junction . . . 91
61 EvacPlan evaluation building ("B2") . . . 94
62 Evacuation progress at 179 ticks and 629 ticks (using Proximity Allocation without delay, "prox") . . . 96
63 Experiment results (evacuation time) . . . 97
64 Temporary deadlock at doorway for 400 agents using prox . . . 97
65 Experiment results (injury counts) . . . 98
66 Evacuation rates for 800-occupant problem . . . 100
67 Rate of injury for evacuation of 800 agents . . . 101
68 Expansion of potential future states for 3 groups meeting in the building 107
69 Orthogonal Paths . . . 107
70 Two crowds occupying the same space but at different times are orthogonal to one another . . . 109

71 Example of CrowdRoute merging. Coloured squares indicate the route the CrowdRoutes are using . . . 113
72 Experiment building with occupants (small circles) and hazard start locations (large, outlined circles) . . . 114
73 Experiment building with graph . . . 114
74 EvacSim running time for 2-1476 agents . . . 115
75 Experiment Building Layout showing initial occupancy (filled circles) and sensor locations (hollow circles) . . . 120
76 Ratio of correct to incorrect scenario matches at individual sensors (average over 10 iterations) . . . 122
77 Average number of correct and incorrect scenario matches by individual sensors (average over 10 iterations) . . . 123
78 Average % confidence in each scenario from Base Station perspective 124
79 EvacSim Evacaution Simulator with agents and simulated sensor network . . . 129
80 Wall Object in Coordinate Space . . . 130
81 Wall Objects enclosing a space . . . 131
82 Line of Sight detection . . . 132
83 Collision detection (Rectangles) . . . 132
84 Doorway Detection . . . 134
85 Room generation . . . 136
86 Generation of Traversability graph from Rooms and Intersections . . 136
87 Traversability Graph . . . 138
88 Example building definition . . . 139
89 Example WiDesign XML building definition . . . 140
90 Conversion from IFC to two-dimensional XML representation . . . 141
91 1-dimensional XMLWalls overlaid on the CoverGrid . . . 142
92 Generation of Walls by filling CoverGrid squares . . . 143
93 Building Draw Tool . . . 144
94 Simulation ticks . . . 145
95 Agent Vector Steering . . . 145
96 Collision correction examples . . . 146
97 Geospatial Indexing . . . 151
98 EvacSim running time for 2-1476 agents . . . 152
99 Example Density/Velocity graph plot . . . 153
100 Example real-time density heatmap . . . 154
101 Voronoi Cell velocity Snapshot . . . 154
102 Averaged Velocity, derived from Voronoi Cells . . . 155

Declaration

All work presented in this thesis is original. I wish acknowledge the contributions of Dr. Tarik Hadzic for his implementation of the EvacPlan planner module and Dr. Alan McGibney for providing XML building definitions. Ths work was carried out under the supervision of Dr. Kenneth N. Brown and Prof. Cormac J. Sreenan between September 2009 and December 2013 in the Department of Computer Science, University College, Cork, Ireland. This dissertation has not been submitted in whole or in part for any other degree, diploma or qualification at any other institution.

Seán Óg Murphy

January 2014

Abstract

Simulation of pedestrian evacuations of smart buildings in emergency is a powerful tool for building analysis, dynamic evacuation planning and real-time response to the evolving state of evacuations. Macroscopic pedestrian models are low-complexity models that are and well suited to algorithmic analysis and planning, but are quite abstract. Microscopic simulation models allow for a high level of simulation detail but can be computationally intensive. By combining micro- and macro- models we can use each to overcome the shortcomings of the other and enable new capability and applications for pedestrian evacuation simulation that would not be possible with either alone. We develop the EvacSim multi-agent pedestrian simulator and procedurally generate macroscopic flow graph models of building space, integrating micro- and macroscopic approaches to simulation of the same emergency space.

By "coupling" flow graph parameters to microscopic simulation results, the graph model captures some of the higher detail and fidelity of the complex microscopic simulation model. The coupled flow graph is used for analysis and prediction of the movement of pedestrians in the microscopic simulation, and investigate the performance of dynamic evacuation planning in simulated emergencies using a variety of strategies for allocation of macroscopic evacuation routes to microscopic pedestrian agents. The predictive capability of the coupled flow graph is exploited for the decomposition of microscopic simulation space into multiple future states in a scalable manner. By simulating multiple future states of the emergency in short time frames, this enables sensing strategy based on simulation scenario pattern matching which we show to achieve fast scenario matching, enabling rich, real-time feedback in emergencies in buildings with meagre sensing capabilities.

Acknowledgements

This work was funded by the NEMBES project, part the Irish Higher Education Authority PRTLI-IV research program, and I wish to thank all my colleagues in the NEMBES project. I would also like to thank the University College Cork Dept. of Computer Science for their financial and administrative support in the final year of this research.

Thanks to ped-net [3] for curating pedestrian movement experiment data used in the validation experiments of Chapter 3 and to Dr. Alan McGibney for providing the XML building files used in Appendix A.2.6.

I would like to thank my friends and family for their support, and my colleagues at the Mobile & Information Systems Laboratory (MISL) and the Cork Constraint Computation Centre (4C) for all our discussions on this research and beyond. I would like to particularly thank to Dr. Tarik Hadzic for his assistance in implementing the EvacPlan planning module, and to Mary Noonan for administrative support throughout this research.

Finally, a special thanks to my Research Supervisors Dr. Ken Brown and Prof. Cormac Sreenan for their advice and guidance.

Publications

Some of this work has already been published in peer-reviewed publications:

[4] S. O. Murphy, K. N. Brown, and C. Sreenan, "The evacsim pedestrian evacuation agent model: development and validation," in Proceedings of the 2013 Summer Computer Simulation Conference, p. 38, Society for Modeling & Simulation International, 2013.
This paper forms Chapter 3 of this thesis.

[5] S. O. Murphy, K. N. Brown, and C. Sreenan, "Problem decomposition for evacuation simulation using network Flow", Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications, pp. 101-108, 2012.
This paper forms the first part of Chapter 6 of this thesis: Section 6.2.

[6] S. O. Murphy, K. N. Brown, and C. Sreenan, "Predictive simulation & evacuation monitoring in wireless sensor networks,", Proceedings of Artificial Intelligence and Cognitive Systems (AICS) 2011, pp. 36-45, 2011
This paper forms the latter part of Chapter 6 of this thesis: Section 6.3.

1 Introduction

1.1 Planning and Simulation of Emergency Evacuation

With the development in recent years of building monitoring and control through the use of networked "Smart building" components (networked sensors, actuation and computing capability embedded within buildings), there arises the scope for real-time management of buildings during emergencies. Traditional emergency response strategies use predetermined evacuation routes based on the building structure but are not in any sense "dynamic"; while a building's sensor network might report on the location of fire or smoke, and provide a real-time perspective of the location of building occupants, static evacuation routes do not account for the dynamic state of building use or hazard spread.

A new class of evacuation planning making use of real-time building state information has developed in the past decade, extending beyond static safety-oriented plans and making use of macroscopic building models (such as flow graphs) to compute evacuation routes for occupants based on their locations in the building (rather than using static routes based on the *likely* positions based on typical building usage). These macroscopic planners can account for dynamic hazard location, redirecting occupants away from danger and towards exits by accounting for distance and population size to avoid congestion. As emergency evacuation is a time-sensitive problem, these planners need to compute routes in very short time frames (less than seconds), which they achieve by modelling the problem space using macroscopic graph models of building space, which are highly scalable and amenable to algorithmic analysis.

While macroscopic planners are capable of computing routes quickly and efficiently, they are quite abstract relative to the realities of pedestrian movement and use of space. Microscopic pedestrian simulations can model pedestrian motion in greater detail, giving a more realistic perspective of the nuances of occupant motion and use of building space. Development of microscopic pedestrian simulation has enabled new applications in building structural analysis, emergency simulation for planning support, and for analysis of the performance of evacuation planning where drills or real-world observation is infeasible or impractical.

1.1.1 Thesis statement

Macroscopic pedestrian models are simple and well suited to algorithmic analysis and planning, but are quite abstract. Microscopic simulation models allow for a high level of simulation detail but can be computationally intensive. By combining micro- and macro- models we can use each to overcome the shortcomings of the other and

enable new capability and applications for pedestrian evacuation simulation that would not be possible with either alone.

1.2 Integration of Micro- and Macroscopic models

In this work we investigate the integration of two models from these classes:

- Microscopic Free-space social force agent model
- Macroscopic Network Flow Graph model

Agent-based microscopic pedestrian models allow for individual occupant models, allowing for a variety of agent mobility characteristics within a single population, or for various behaviours or building knowledge to be modelled on an individual agent basis. By modelling agent motion in continuous free space, multi-agent pedestrian models can capture much of the nuance of occupant use of building space, and the impact of different agents interacting in space. The flexibility (allowing multiple agent types to exist within the same model) and realism (allowing for complex motion models) make this approach to pedestrian modelling particularly suitable for emergency simulation; however this approach to simulation is complex and scaling of simulation problems is a challenge.

Network Flow Graph models are particularly suited to evacuation planning problems as they model building space in terms of not only of how space can be traversed, but also the limitations of space when it comes to the rate of "flow" possible through the space. As congestion and over-use of paths by occupants is a significant component to building evacuation and safety, accounting for the achievable flow is key to safe evacuation planning and this accounting for flow combined with their inherent scalability and amenability to analysis makes network flow graphs particularly suitable for planning and prediction of the traversal of groups of occupants.

By integrating these two models, we can use the microscopic agent model to improve the accuracy of the macroscopic graph model, imparting some of the subtle detail of occupant use of space to the more abstract graph model, allowing for prediction and planning using the Network Flow Graph based on microscopic simulation. Integration also allows the microscopic agent model to exploit the analytical powers of the macroscopic flow model to sub-divide problem space and achieve scalable microscopic simulation of multiple future states, which would otherwise be computationally infeasible within the time constraints of real-time emergency response and simulation.

1.2.1 Macroscopic Pedestrian Modelling

Macroscopic Models used in pedestrian movement models are typically graph-based, and in this work we make extensive use of Network Flow Graphs as macroscopic representations of building emergencies. Network Flow Graphs model building space in the form of a graph of nodes and edges, and each edge has Capacity and Traversal Time parameters. These parameters represent the time taken for occupants to traverse the edge, and the optimum rate of traversal (or "flow") achievable on that edge. In these graphs, pedestrians exist as "supply": the number of occupants present at a node. As such, individuals are not considered, but rather the transfer of this supply from node to node via edges models the traversal of occupants in the building space. When combined with occupancy and hazard data, Flow Graph models can be used by dynamic evacuation planners as a basis for generation of evacuation plans. While Flow Graphs are computationally simple, they are also quite abstracted from the physical domain. The nuance of interaction of occupants with building geometry, or the impact of multiple flows of occupants coming together or passing through common space can be complex and using graph models alone can miss out on this fine detail, which can have a significant bearing on the accuracy of graph-based simulation, prediction or planning.

1.2.2 Microscopic Pedestrian Modelling

Microscopic Models of pedestrian movement are much more detailed, typically modelling each occupant individually as "agents" in building space, with each agent responsible for its own decision making, planning and movement as they navigate the building geometry. A common approach for occupant modelling in microscopic pedestrian simulation is to adopt a "social forces" model of agent interaction, which we use in this work, treating agents as particles in two-dimensional free space with agent positions adjusted based on the forces model each time the simulation is updated. As each agent is modelled individually, this allows the persistence of state; agents can plan individually, and extensions to agent models can allow for the capability of learning, information sharing, emotional states and personality types. This also allows for the integration of different agent types within a single simulation, occupants with limited mobility or different knowledge of the building structure can exist together within the same model.

In microscopic social forces models, the motion of agents is governed by a combination of forces: maintaining personal space, movement towards a goal, adjusting motion to avoid collisions and steering to navigate space. Each update of the simulation causes the agents to modify their position in order to satisfy the forces; for instance by making small adjustments they can move to positions with greater space

to maintain personal space needs, and by steering and moving towards a series of goals in sequence, they can navigate the building space. Microscopic models are detailed, but complex. While graph model complexity scales with the size of the graph, microscopic social forces models scale with the number and density of agents modelled. Large crowds of agents result in a great deal of inter-agent force computation. The high detail and fidelity of microscopic models also present a challenge when it comes to agent planning; agents need to plan movement over long distance between distinct goals, often obscured by building geometry.

1.3 Integration

By using macroscopic graphs and microscopic free space agent models to represent the same physical space (e.g. a building), we can allow one model type to make use of the capabilities of the other to achieve simulation and modelling capability which would not be possible with either alone. A common approach to agent path planning in microscopic agent simulation is to allow the agents to perform high-level planning with the use of a "navigation graph", a graph representing the traversable edges between building spaces. By associating areas in the building with nodes on the navigation graph, agents can perform path-planning through the building using algorithms such as Dijktra's Algorithm [7] or A* [8].

Microscopic modelling can also assist macroscopic models; by using social forces agent modelling to simulate the movement of pedestrians through space, the rate and speed of traversal can be discovered through that space and this can be used to establish Network Flow Graph parameters automatically. By using the microscopic simulation to iteratively test flows on point-to-point paths represented by graph edges in the Flow Graph, the Flow Graph can be "coupled" to the microscopic model, allowing the abstract macroscopic graph model to reflect some of the more detailed nuance achievable in the microscopic free space simulation, without the need for manual assignment of flow parameters.

With microscopic and macroscopic models coupled together, we can use the Flow Graph as a predictive tool for estimating the likely outcome of microscopic simulation prior to actually simulating it. As macroscopic graph models are abstract and do not model the interaction of groups as effectively or realistically as free space, social forces agent models, the simulation of interactions in microscopic context is still required in order to achieve an accurate simulation. By analysing the simulation world using the macroscopic graph, the near-future locations and participants in these group interactions (which macroscopic simulation handles poorly) can be identified ahead of time. By identifying these cases, we can determine which microscopic agents will interact with each other in the near future of the simulation, and

which are not likely to meet each-other, allowing for the high-detail, high-complexity microscopic simulation to treat occupants that are predicted to meet separately to occupants that are not predicted to ever collide in space. Through this analysis we can significantly reduce the complexity of microscopic simulation by subdividing the simulation world based on this orthogonal relationship in time and space.

1.4 Outline

The structure of this work is as follows:
in **Chapter 2** we investigate the related work to micro- and macroscopic pedestrian simulation; we describe microscopic and macroscopic pedestrian models and identify some examples of hierarchical and hybrid approaches that make use of combinations of these model types. We describe some real-world pedestrian experiments and data, and approaches to geometry analysis that enable the translation of microscopic representations to macroscopic models. We also describe evacuation planning approaches that exploit macroscopic models for dynamic generation of evacuation plans and some methods of sensing and actuation in buildings with networked sensors and communication.

Chapter 3 details the microscopic agent movement model used by the EvacSim Pedestrian Evacuation Simulator (Appendix A). We describe the parameters governing the agent movement moment-to-moment during simulation and evaluate the accuracy of this model against real-world results from literature documenting the characteristics of pedestrian movement through spaces.

Chapter 4 details the extraction of multiple levels of macroscopic graph detail from the basic graph generated by EvacSim. Each graph detail level is appropriate for different purposes; high density graphs with many nodes and edges allow for a great deal of choice and control in planning for microscopic agents navigating space, while sparse graphs with nodes representing large spaces are more suitable for macroscopic flow-based planning. These graph details are arranged hierarchically, which allows for paths and features in one graph level to be translated to others, allowing for high-level macroscopic planning to be communicated to microscopic agents.

Chapter 5 describes the "coupling" approach in which the EvacSim microscopic simulation is used to establish the parameters of the macroscopic high-level flow graphs. By automated coupling of the graph to the free space simulation, it can be used for prediction of microscopic simulation performance: the time taken for occupants to traverse space can be predicted ahead of time. We investigate the predictive capabilities of the graph when coupled to microscopic simulation, and investigate the performance of a dynamic evacuation planner during simulated emergencies when using coupled flow graphs and a variety of occupant densities and path allocation

strategies.

In **Chapter 6** we make use of coupled flow graphs as analytical tools to subdivide simulation space and allow for scalable simulation of branching future states. By analysing the graph, the future interactions (or lack thereof) of microscopic agents is predicted and these predictions used to reduce the size of simulation instances. We also describe a method of exploiting these future state simulations by generating simulation "sensor fingerprints", which, through real-time pattern matching, can be used by networked sensors in the real building to identify which of many previously simulated future states matches the best with their sensor perspective.

2 Related Work

2.1 Introduction

In this Chapter, we describe the state of research in Emergency Management, Emergency Simulation, Real-time simulation and Evacuation Planning. We describe approaches to feature detection for discovery of building topology, and the use of Topological and Flow graphs to represent the traversal characteristics of the building, the foundation of the Macroscopic modelling approaches used in this research. We also describe related Microscopic pedestrian simulation, and describe their relationship to the model used the EvacSim simulation. Validation of this microscopic model in Chapter 3 makes use of results from a number of real-world pedestrian movement experiments performed by other researchers which are also described here.

We begin with some discussion on Building Evacuation in Smart Buildings. Buildings with sensors and computing capability motivate much of the real-time emergency evacuation planning and simulation approaches described in this thesis. Following this, we describe some Building Models and approaches for analysing building geometries to extract macroscopic features. For microscopic and macroscopic models to augment each other, they need to model the same space. Using Feature Detection algorithms, free-space building models can be converted to macroscopic graphs, and we describe some approaches to feature detection in Section 2.3.1.

Macroscopic graphs can be used for emergency simulation and also for emergency planning, and we discuss some releated work on this in Section 2.4. Macroscopic planners produce dynamic evacuation plans for emergency situations that are responsive to the state of the emergency, and can be communicated to occupants either directly (such as to smart phones or personal guidance) or through building actuation (dynamic signposting) as part of the communication network of Smart Buildings. Graph models are also useful for agent path planning, and we describe some approaches to graph modification and planning in Section 2.4.2.

Following this we describe some related microscopic pedestrian simulation approaches. Microscopic simulation allows for high-detail simulation of pedestrian movement and this makes it well-suited to modelling emergency situations where the interaction of individuals in space has a substantial bearing on the outcome on safety. A particular strength of agent-based approaches is that they allow for individual agent behaviours, and we describe some behaviour and cognition approaches in Section 2.5.2.

From here, we describe approaches to simulation where different model detail levels are combined: hierarchical models. By using different model details for different parts of simulation problems, the computation resources can be focussed the most

significant, interesting or dynamic parts of the simulation world.

Finally we describe real-world pedestrian movement data, which enables the validation of pedestrian motion models in Chapter 3 and give context to emergency simulation with some examples of hazard modelling.

2.2 Sensing in Smart Buildings

Integration of networks in buildings allows for real-time monitoring of building state, of particular interest in emergency situations is the monitoring of occupancy (location of occupants) and hazard (identifying areas that are hazardous; fire, smoke etc). Deployment of Sensors enables the collection of this monitoring data, and by networking the sensors together the data can be collected at a central location and exploited for monitoring and planning. Another important aspect of smart building sensor networks is the facility for actuation; the communication of instructions to network components such as dynamic signposting, or communication of guidance directly to occupants using networks smart devices such a cellphones. A common approach to the deployment of sensor networks in buildings is the use of a "Wireless Sensor Network" or WSN; these networks are composed of multiple individual wireless nodes which are typically battery powered and deployed without the need for wiring. These WSNs are particularly interesting for emergency situations as the lack of wiring means they can be more robust to damage and by dynamically adjusting routing tables and schedules they can adjust to loss of nodes due to damage and maintain communication even with the loss of several nodes.

In recent years interest in tracking individuals using wireless sensor networks has increased, much of this interest coming from the security industry, target tracking with WSNs is potentially a valuable capability for security systems and location-sensing is a critical component for future augmented-reality/pervasive computing platforms. Target tracking can take the form of untagged tracking in which the individual under surveillance does not carry any identifying item (via cameras, infrared sensors, pressure and movement sensing) or tracking of individuals who are identified or tagged in some way (via passive or active RFID tags, mobile wireless devices in real-time communication with the WSN, etc.).

Amin [9] developed a sensor pattern recognition approach that parallelises pattern recognition by distributing the task among multiple sensor nodes, by treating the networked nodes as a Neural Network. This approach allows for fast recognition of patterns by combining several sensor readings and using a single-cycle learning algorithm. This approach to sensing is similar to the pattern-matching strategy employed in Section 6.3.

Goel et al.'s work on Predictive Monitoring (PREMON) [10] demonstrated the en-

ergy savings possible when future sensor readings are predicted centrally, and they show benefits of exploiting correlation between readings on different sensors while minimizing work performed by the sensors. PREMON is motivated by the desire to reduce the energy consumption of wireless nodes and utilises an MPEG-style encoding scheme which uses a global view of sensor reading history to predict the next global set of sensor data. The predicted values are sent to sensors and the sensors need only report back to the Base Station if the predicted values mismatch with the real readings. This approach predicts future readings based on previous sensor readings (as frames in an MPEG-style movie). However, PREMON generates only the most a single likely future prediction. In scenarios with multiple tracking targets, such as a building evacuation, there are many possible future events, and there is a low possibility of being able to predict the true future event. Predicting the wrong event, and thus actuating the wrong guidance, could have a serious negative effect.

The Dual Prediction-based Reporting (DPR) mechanism developed by Xu et al [11] operates using duplicate predictive models running on the base station and on individual sensor nodes as a basis for single-target tracking. Each node uses the model to predict future sensor readings based on historical data. As the base station possesses duplicates of these predictive models, the sensors need only inform the BS of their readings in the event that the predictive model failed to match with the observed readings. The predictive model is based on linear-projection of previous target positions and sensors share their historical readings with neighbours to facilitate the tracking of the target as they move from one sensor field into another. DPR demonstrates significant energy savings over constant monitoring and PREMON but suffers from the poor scalability of its predictive model. Tracking of large numbers of individuals requires a large set of predictive models at each sensor node to predict future positions of each individual. DPR also requires that the sensors are capable of maintaining knowledge of the identity of individuals in the building: the predictive model predicts future movement of an individual based on previous known positions of that particular individual.

Pattem et al [12] demonstrated the energy benefits of using target position prediction to schedule wakeup of tracking nodes. In this work, they utilised a basic single-target tracking algorithm which predicts the targets future position as a linear projection of their observed path and movement speed. They present the results of a series of simulations which demonstrate significant improvements in energy usage when prediction is used to wake up nodes, as compared to naïve activation (all nodes awake) and random activation (nodes wake up according to a randomized schedule). The reduction in tracking accuracy due to this predictive-wakeup strategy is compared to naïve activation (which represents the best tracking possible as all nodes are in tracking mode at all times) and the authors conclude that selective activation based on prediction provides near-optimal tracking with substantial en-

ergy savings. As can be expected, the quality of target tracking is relative to the quality of information regarding tracking node locations.

Souza et al [13] compared the performance of a single-target tracking algorithm for WSNs in which the sensor location information was discovered using a beacon-based localization algorithms. These localization algorithms require a small number (<5%) of sensors with known location to act as beacons and the remaining sensors resolve their location based on communication information with these beacons and each other. The target tracking algorithms used were based on Kalman and Particle filters. The authors present a series of experiments simulating different combinations of localization and tracking algorithms, deployments densities and network scales. They conclude that the tracking algorithms used in this work were unable to filter out any error that may have been introduced in the localization algorithms and that the combination of tracking and localization techniques is relatively unexplored and further research may lead to improvements in both areas.

Balister et al [14] propose a WSN tracking network model called Trap Coverage which allows for deployments with gaps in sensor coverage (holes) while maintaining a guarantee that targets moving above a particular threshold distance are detected. This model improves on prior full coverage models by reducing the total number of sensors required for deterministic guarantees of detection over an area and the authors conclude that deployments based on Trap Coverage rather than Full Coverage would scale better than full coverage for large deployments. This approach identifies critical areas in the tracking area that are sufficient to provide full tracking coverage, while also identifying areas which do not need coverage as they do not aid in distinguishing one event from another.

Sanli et al [15] developed a tracking strategy which attempts to maintain tracking of multiple targets by maintaining an identity for each target and using probabilistic models based on observed movement trails to preserve this identity when multiple targets enter a common space ("mixing regions") where tracking may be ambiguous. By observing target movements over time, the tracking system learns the preferred trails of individual targets, so that when multiple targets enter a mixing region simultaneously, the system predicts that each target follows their usual preferred trail. The authors present tracking experiment results involving 2–7 targets and 150–350 tracking nodes comparing this with the Multiple Hypothesis Tracking algorithm presented by Blackman [16], noting substantial improvements in energy usage and tracking accuracy, particular as the number of targets under surveillance is increased.

Relation to this work

Building sensing and aggregation approaches allow for sensor learning, coverage and duty scheduling that increase the capabilities of sensors beyond the sum of their parts. However, learning and prediction is contingent on there being data

to learn from and typical building sensing does not often have the opportunity to observe building emergencies. Furthermore, local sensor aggregation and learning approaches are at risk in emergencies, where fire or sensor energy death may result in the loss of critical nodes in the sensor network or disconnection of the WSN from central computational resource ("Base Stations").

By implementing highly scalable multiple future state simulation, we can simulate many different future states of building evacuation and extract simulated sensor fingerprints from each of these. While the approaches to sensing discussed here exploit the aggregation of sensor readings for learning or coverage, microscopic simulation of future states could provide for models to match sensor readings with, or as the basis for sensor wakeup cycles such as for [12] (where the sensor wakeup schedule is based on the expectation of important events to sense). While individual sensor capability may be limited, by associating the high detail of simulated scenarios with the low complexity of basic sensor-level pattern matching, this enables real-time response to relevent changes in the building evacuation even with very meagre sensor capabilities, which we discuss in Section 6.3.

2.3 Building models

When considering emergencies in buildings, a model of the building is required. While it is possible to produce bespoke modelling types, it is useful to consider the existence of a variety of general-purpose building modelling options. Traditional building blueprint-style models have evolved into complex 3-dimensional building drawings created in 3D drawing tools such as ArchiCAD [17] or Revit [18], and it is increasingly common that these models are available for public buildings or other large facilities. An interesting development in recent years is the development of the Industry Foundation Classes (IFC) [19] building modelling language; a markup language for hierarchical description of building geometry and materials designed to enable interoperability of different Building Information Modelling (BIM) tools. With respect to Smart Buildings, IFC allows for the representation of building network components within the building model itself, allowing for a single file format to represent both the materials and geometry of a building and can be extended to represent the building's network components, actuation and sensing capabilities. IFC can be dynamically updated throughout the life of the building, and can provide a real-time perspective of the status of building Networks and can reflect modifications to the network (such as replacement of nodes or modification of sensor components). IFC definitions have been used as the basis for building energy use simulation [20] [21] [19], and is a supported format for importation of building geometries to the EvacSim Simulator used in this work (after conversion to XML, described in Appendix A.2.6).

2.3.1 Feature Detection

Binary Space Partition (BSP) is the recursive subdivision of space by dividing space at convex corners to produce a hierarchical tree where leaf nodes are convex polygons, a BSP Tree (Figure 1. BSP trees provide an efficient hierarchical representation but the space division can result in many irregular shapes that have little in common with occupant utilisation of space.

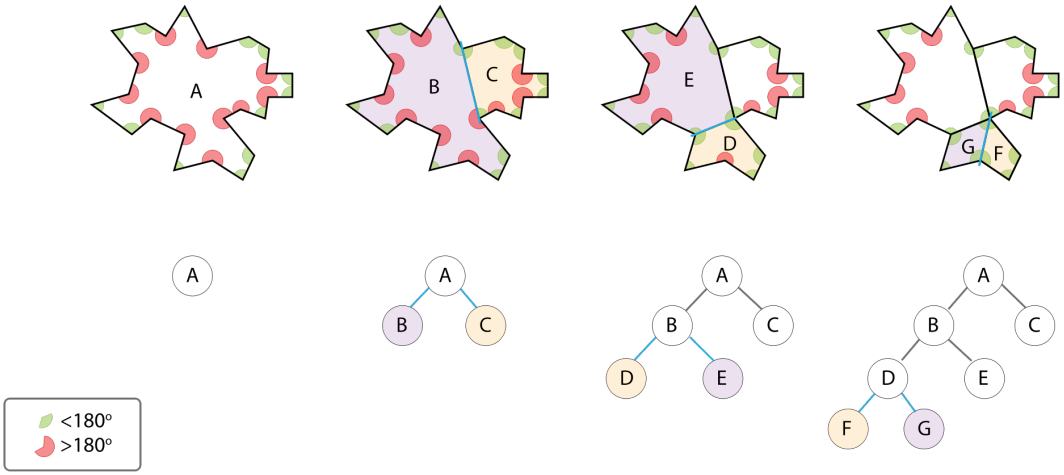


Figure 1: Binary Space Partition and BSP Tree

Lerner [22] proposed an alternative to BSP called "Breaking the Walls", an algorithm that iteratively divides the space into discrete cells by walking along the surface of three-dimensional walls in the world, jumping gaps if they exist and only ever turning in one direction. When the walk arrives at a position it has already visited, the enclosure generated can be saved as a new cell. Portals (the surface common to both cells) are used to reduce the computational complexity of line-of-sight calculations, for graphics rendering pruning problems. The portal serves as the basis for a viewing frustrum from a camera in one cell to objects in adjacent cells, and objects lying in those cells but outside of this frustrum can be culled (and not rendered).

Lerner's approach is an efficient method of space subdivision, and if the layout of obstacles is aligned to the horizontal and veritcal axes, produces results very similar to that of the Rectangular subdivision used in EvacSim (Appendix A.2.4). In the case of more irregular obstacle geometries, this approach achieves an orderly subdivision but creates irregular polygon cells which makes localisation of objects in the simulation world more difficult than using simple rectangles as spaces.

The "Watershed" approach proposed by Haumont et al [23] identifies rooms and the intersections between them by simulating the rising level of water in a three-dimensional gradient based on the position of physical obstacles (sloping upwards from the centre of a room to the walls). By iteratively raising the level of water,

large areas (at the bottom of the gradient) are flooded first, and eventually spill into each other as the water rises, at which point an intersection has been discovered and a gateway portal can be placed at the point of intersection. This approach tends to divide space at doorways, allowing for irregular rooms to be represented by a single object, but these irregular shapes are problematic for object localization; for instance an L-shaped room would be composed of five vertices as a concave polygon, rather than two distinct convex rectangular boxes.

Relation to this work

The microscopic simulator used in this work (EvacSim) supports IFC building definitions converted to XML, allowing for the support of a number of building model types through a conversion toolchain (such as from ArchiCAD to Revit IFC file, and from this to XML and finally to EvacSim building file). To divide space in a logical manner, and produce topological graphs of building space, EvacSim uses a space division method based on expansion of rectangles to identify room spaces, doorways and areas of adjacency between these spaces and doorways (Appendix A.2.4). This approach was taken as it matches well the with the goals of EvacSim of low computational overhead as rectangular spaces simplify the geospatial localisation of objects in the simulation world.

2.4 Macroscopic Models

Macroscopic modelling of emergencies takes an abstracted perspective; macroscopic approaches are highly scalable and algorithmic analysis of graph problems is the foundation of a new class of dynamic evacuation planners which attempt to plan evacuation routes for occupants in emergencies based on the dynamic state of emergencies. Macroscopic graph-based models have also been used as the basis for macroscopic building evacuation simulators.

2.4.1 Macroscopic Emergency Simulation

In this work, we use topological graphs to model the traversability of building spaces, and augment these graphs using flow parameters to produce Network Flow Graphs which can be used for prediction and dynamic evacuation planning. Filippoupolitis et al [24] [25] [26] make use of a directed network graph as a planning tool, and also as a simulation approach in the design and evaluation of a safest-path evacuation planner. By locating the hazard in the building, the set of edges in the graph are directed so that they point away from the hazard and provide a shortest path away from the hazard and minimise exposure. A limitation of this work is that the use of a graph-based simulation as a validation for the planner which uses the same kind of graph model fails to identify flaws in planning that arise from the difference between

the model used and more realisitc pedestrian behaviour. In Chapter 5 we found that using a more detailed, realistic simulation model to evaluate the performance of Flow Graph-based prediction and planning identified a number of flaws with the graph-based modelling approach which would not have been identified were these graph models evaluated using graph-based simulation.

While Cellular Automata (CA) Section 2.5.1) is typically considered a microscopic simulation approach, it has much in common with macroscopic coarse graph models: graphs composed of dense grids of nodes each connected with up to 8 neighbours. The weight of these edges is given by the amount of time taken to traverse the distance, which overcomes the limitation of CA models in accurately representing the different in traversal time between up/down movement and diagonal movement. The coarse graph model is one of the approaches used in buildingExodus [27] and is also a submodel in the hybrid simulation approach used by Chooramun [28].

2.4.2 Graphs and Path-Planning

Graph models (composed of Nodes connected by Edges) are macroscopic models that can be created to represent the traversability of space. By creating a topological graph representing this traversability, it is possible to algorithmically discover routes (in terms of nodes visited, or edges traversed) between different points of the graph (and thus discover routes between different parts of the simulation world). In 2.3.1 we look at some approaches to the creation of these graphs, and in Chapter 4 we describe the approach used in EvacSim, which allows for the generation of multiple graphs of different details to represent the traversability of building space. By assigning capacity values to edges in typical topological graphs, we can extend the model into a "Flow Graph" [29], which not only describes the traversability of space (and the time taken to traverse it), but also represents the maximum number of people that could traverse the edge simultaneously, capturing the limitations of space when utilised by a number of occupants.

Richter [30] describes use of a hierarchical tree for representing the hierarchy of building spaces; for instance an Office is part of a Department, which is part of a Floor, which is part of a Building. The goal of this is to allow for high-level route descriptions, for example: Turn right, enter Floor B2, then get over the stairs. Pass Civil Engineering offices and turn left at the postgraduate lab to find Room 205. As part of the Hierarchical Level of Detail graph generation in Chapter 4, we show the production of high-level graphs that amalgamate spaces between doorways, allowing for the description of complex route (from higher-detail graphs) in terms of the simpler structure of the high-level graph, as in Richter's work.

Erikson [31] described an approach to simplifying 3D geometry to allow for varying

levels of details (depending on distance from viewer in scene). While this approach is used for 3D model simplification, Erikson's vertex collapse approach to simplification has similarities to the Loop- and Triangle-Removal approaches we adopt for Topology Graph simplification described in Chapter 4. Vertex collapse is used to merge vertices producing progressively simpler models, merging nodes that are near together in space and hence have a relatively low impact on the overall quality of the model. The merged nodes can be associated with parent nodes in a balanced Edge Heap, producing a hierarchy of detail levels for a single 3D object. When viewing large objects, the nearer parts of the models go deeper down the tree, showing more vertices allowing a dynamic scene complexity depending on the position of the viewer.

For individual planning problems, the major class of relevant algorithms are weighted graph Shortest-Path algorithms. These algorithms find paths through weighted graphs from one node to another and attempt to minimise the total distance (sum of edge weights) of the path found. When used for path planning in two-dimensional space, graph nodes are associated with positions in the space, and the edge weights represent the cost of traversal between two nodes (typically the distance between the nodes), Some algorithms make use of heuristics to reduce the search space, typically using straight-line distance between node positions to direct the search in the approximate direction of the goal, and this is the approach for individual agent path-planning used in this research. We also highlight here some probabilistic planning approaches and algorithms that are well-suited to partial replanning in response to changes in environment (such as addition or removal of graph nodes or edges).

Dijkstra's Algorithm [7] is the classic shortest-path discovery algorithm, which operates by iteratively exploring the graph from the starting node and noting the shortest path to each node as it is discovered. As Dijkstra's algorithm continues the exploration from the shortest path so far plus the shortest next edge, it guarantees that it will find the shortest path provided that there are no negative weight edges.

A* is a common approach to shortest-path discovery that greatly improves computation time compared to Dijkstra's Algorithm by the use of heuristics to direct the exploration. By combining the path length value with a heuristic (typically the straight-line distance from a node to the target), the exploration of nodes can be directed in the approximate direction of the goal and this typically achieves substantial savings in computation time for problems where an admissible heuristic exists. A* is the algorithm used for general agent path planning in this work, with straight-line distance acting as the heuristic and computed based on cartesian distance between the centrepoint coordinates of nodes.

Sud et al [32] give a survey of their work on path planning in real-time in simulations

of multi-agent systems. They approach multi-agent crowd simulation as an application in computer game crowd simulation and other virtual environments such as avatar-based social networks. With this approach in mind, they are focused more on pleasing or realistic appearance for users of such virtual reality environments rather than a close correspondence to real life crowd scenarios. An important element of crowd simulation in videogames is that navigation and planning calculations need to be performed in real time so as not to impact on the performance of the game (i.e. to avoid unpleasant slowdown by maintaining a high update rate). They present an approach to reducing path planning computation time by sharing a navigation graph among all agents (Multi-agent navigation graph) and state that this approach can support path planning for several hundred agents in real time. They also present a technique for reducing complexity by supporting varying levels of detail (pedestrian level of detail) which allow accelerated simulation by clustering agents together and performing calculations using this cluster.

2.4.3 Dynamic Evacuation Planning and Guidance

Development of building sensor networks has resulted in the rise of planning algorithms that are called at the time of emergency and account for the current state of the building, which are termed "Dynamic Evacuation Planners" as they plan based on the currently observed state of the building rather than using a fixed model. Unlike static building evacuation plans, these planners exploit sensor data giving the occupancy state of buildings or the location of building hazards like fire and smoke to produce evacuation strategies specific to the particulars of each particular emergency. While traditional pre-defined building evacuation plans (as are used in many large public buildings) are designed to achieve clearance of buildings in a short time, they lack any facility to respond to the different state of the building during emergency. Predefined plans may route occupants through areas that are on fire or smoky, and do not account for the location of the occupants. Without considering the occupancy of space, these plans could result in an excessive amount of time taken to clear the building if an unexpected number of occupants are instructed to use a relatively narrow, low-capacity space, resulting in congestion and potential delays or injuries.

Dynamic evacuation planners account for the location of the hazard, or occupants (or both) and produce evacuation plans that attempt to minimize exposure to hazards while also minimizing the time taken for full evacuation of the building. The intent of these planners is to communicate the dynamically generated plans to the occupants, though this area of research has been largely theoretical and the task of communication of plans to occupants is not generally considered.

The Graph-based Real-Time emergency evacuation planner developed by Hadzic [33]

is implemented as the EvacPlan planning module used in Chapter 5. This algorithm plans evacuation routes using a flow graph (a topological graph of the building with restricted flow capacity parameters on graph edges). With building occupancy state (as "supply" on nodes) and the known location of hazards, this planner computes evacuation routes for occupants while considering the limited capacity of building space (as capacity of edges). By considering both the traversal time of graph edges, and their limited capacity, this planner is "congestion aware" and uses heuristics (based on exposure to danger and the length of the path) and to produce evacuation routes that clear the building quickly by distributing the pedestrian traffic across multiple routes. The planner computes very quickly, and is compared favourably with the CCRP [34] heuristic flow-based planner on evacuation time and safety metrics.

Filippoupolitis' [24]and Tabirca's [35] planners were also developed with sensor networks in mind. These approaches both consider evacuation based on the criteria of safety and distance from danger. Unlike Flow-based planners, these approaches are based on graph directed edges, and while they are dynamically responsive to hazard location, do not make consideration of crowd congestion. While they translate well to some forms of emergency guidance (directed edges can be expressed easily through actuated signposts, for example), they do not consider the volume of pedestrian traffic being directed through areas of the building and the delays of injuries this may cause.

For occupants to receive and obey evacuation plans, the plans need to be communicated somehow. Chih-Yung Chen [36] described fuzzy planning for the actuation of dynamic signposting in emergencies. These signposts feature directed arrows which can be lit up to point in the direction crowds should travel. Ferscha et al. [37] described the "Lifebelt", a haptic-feedback belt to be worn by occupants which vibrates to direct the wearer in the direction they need to travel, and developed a cognitive agent model [38] which models behaviour of occupants that use the Lifebelt. The belt is worn around the waist and features a set of vibrating panels. These agents are modelled as Cellular Automata (Section 2.5.1) and feature information sharing capabilities and behaviour based on an "emotional state", with agents experiencing panic behaving differently to calm agents.

Chittaro's [39] approach to communication of routes is to present the user with a 3D arrow and map, sent to a portable device, in a manner similar to 3D GPS maps for vehicles. The 3D guidance "snaps" to the direction the occupant is facing, presenting a graphical representation of the space the occupant inhabits on a screen. Occupants are tracked using active RFID tagging of building space, and given that the guidance is provided via a portable smart device with RFID capability, this allows for real-time localisation of the occupant without centralised tracking or monitoring (provided

that the device possesses a map of the building space and tags).

Relation to this work

Dynamic Evacuation Planning using macroscopic graphs is in part, contingent on the quality of the graph models. In Chapter 5 we algorithmically generate flow graph parameters for use in dynamic evacuation planning, allowing the computationally simple flow graphs to reflect some of the higher detail of pedestrian motion in building spaces that would be difficult to capture through manual flow parameterisation. We investigate evacuation plan allocation strategies for the communication of macroscopic graph routes to microscopic pedestrian agents, and identify some limitations of dynamic evacuation planning when guidance instrumentation is limited. The challenges of communication of personalised evacuation routes to occupants and the uncertainty of their following them as the planners expect motivates the goal of scalable microscopic simulation to produce feedback support to planners. By simulating evacuations in faster-than-real-time, the microscopic simulation is capable of informing planners about flaws in their planning strategies within a time-frame that would allow for replanning to compensate. Additionally, by simulating many future states that the evacuation could end up in, ahead of time, we can compute contingency evacuation plans before the contingency occurs, and make these available to building actuation and sensing infrastructure (Section 6.3).

2.5 Microscopic Pedestrian Simulation

Multi-agent real-time simulations of populations of individuals have been developed in recent years as a tool for simulating crowds in a variety of scenarios as a tool to simulate emergency evacuations, recreate historical settings, automate character behaviour in entertainment (movies, videogames) or evaluate emergency response procedures or building architecture. Current research investigates a wide variety of modelling schemes to represent buildings and occupants, with the great majority of this research involving two-dimensional floor plans rather than potentially more complex 3-dimensional models. The agents in the simulations may have varying models of behaviour governing their actions such as planning algorithms or finite state machines, Tavars and Galea [40] note that there exist over 40 evacuation models with different building models and occupant characteristics and that evacuation simulations using multi-agent systems are a valuable complement to mathematical models of building egress, providing a greater degree of accuracy in scenarios where large crowds and congestion can occur.

2.5.1 Microscopic Agent Models

The typical approach to Microscopic pedestrian simulation is to model occupants as particles in two-dimensional free space, and to treat the relationship between the occupants and the simulation world as a system of forces acting on the agents. Helbing [41] introduced the Social Forces approach to agent modelling, a system of equations periodically updated which adjust the agent position in free space based on the agents desired movement, proximity to obstacles and with other agents. These forces include the desire to maintain personal space around agents, intent to travel to a particular destination, inertia, rate of deceleration and so on. The Social Forces approach has contributed to the development of a variety of microscopic pedestrian simulations, and forms the basis of the agent model used in EvacSim.

Wagoum et. al [42] demonstrated a microscopic simulation subdivided into discrete sub-simulations in order to distribute simulation tasks among parallel computation resources. By dividing the space in a large soccer stadium based on gateway or portal areas (bottleneck areas that experience simple, unidirectional flows of traffic), the overhead of communication between computation nodes is minimized. Limitations of this approach are that the space between these gateway areas must still be simulated as a single instance (i.e. large common areas cannot be subdivided), and in this work the transfer of occupants from one simulated area to another is ignored (instead, an estimated rate of transfer is used to generate new occupants at the gateways, rather than transferring existing occupants from one instance to another).

Sharma's model [43] combined Social Forces with fuzzy emotional reasoning for agents, an approach that enables evaluation of evacuation based on the impact on the emotional state of the occupants; allowing for the modelling of stress and panic in response to evacuation events such as exposure to smoke and overcrowding. Agent observations of the environment are used as input for the emotional reasoning, with increased exposure to stressful events resulting in an increase in stress-induced behaviour.

Tavars and Galea [40] present an approach to building evacuation analysis combining occupant simulation (with basic behaviour) with numerical optimisation techniques. The authors conducted an experiment in which the location of a doorway in a simple building geometry was adjusted between iterations of the evacuation simulation and the average evacuation time for a population of 200 agents was recorded for each iteration. The position of the doorway is adjust using a conjugate gradient algorithm (Fletcher-Reeves method [44]) to optimise for evacuation speed.

The buildingExodus [27] simulation suite is an ongoing research project designed to simulate pedestrian behaviour in emergencies. buildingExodus implements a Social Forces model for pedestrian model along with a number of macroscopic models (in-

cluding coarse and fine graph models) and the parallelisation of the simulation space is described in Yasmina Mohedeen's doctoral thesis [45], in which a combination of geospatial indexing and graph analysis is used to divide the simulation space in a manner that balances the simulation load of each subdivision.

The "flocking" approach to agent modelling described by Reynolds [46] was an inspiration for some of the techniques used in the EvacSim agent model described in this thesis, such as "stream formation" (Section 3.7). Flocking delegates responsibility for agent movement to the individual agents, which then observe the positions of the other agents around them and use these to modify their positions, with emergent crowd movement arising naturally as a result. The main benefit of flocking approaches is that they allow for individual agent rulesets to dictate movement as distributed, heterogeneous behaviour rather than global-level rulesets like Cellular-Automata (Section 2.5.1) or Graph Models (Section 2.4).

Olfati-Saber [47] presented a framework for designing and evaluating crowd-simulation using a distributed flocking algorithm. The author identifies shortcomings of prior flocking algorithms which make assumptions that lead to undesirable features such as a fragmentation or flock collapse, and lack distributed migration ability for crowds, and seeks to overcome these shortcomings with a distributed flocking algorithm. An interesting contribution from Olfati's research is the description of a peer-to-peer (local agents as peers) distributed obstacle avoidance scheme by modelling obstacles as a special case of flocking agent (β agents). This flocking strategy is evaluated based on the degree of fragmentation of flocks (based on deviation of a triangular/polyhedral lattice of agents) when manoeuvring around obstacles.

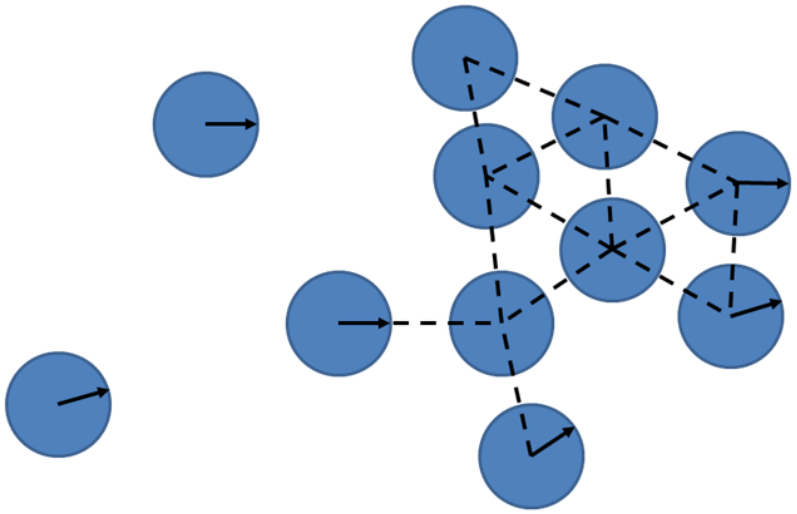


Figure 2: Flocking agents forming an α-lattice

The author also described an extension to flocking which allows for collective planning, through the creation of γ -agents (where individuals are termed α agents, and physical obstacles are termed β-agents). γ-agents are generated when groups

of α-agents are adjacent forming cohesive groups (forming an α-lattice, Figure 2). The members of this α-lattice can be grouped together and γ-agents generatted to govern the decision-making process of the crowd as a whole.

Cellular Automata (CA) is a common alternative to social forces/flocking-style approaches, that simplifies agent movement by modelling the simulation world as a network of "cells" with agents occupying these cells and a set of rules governing the movement of occupants between these cells. This approach is relatively simple in computational terms, as it models movement from cell-to-cell in a discrete grid, with simple rules such as "no more than one occupant per cell" and "movement only allowed between adjacent cells" dictating possible movement. While CA is simple to implement and compute, it is also significantly more abstract than free space social force models and are of limited utility as models for investigating congestion. Examples of Cellular Automata Pedestrian Evacuation simulation include Borrmann, Kneidl et al [48], Schadschneider et al [49], Klüpfel [50] and Ji et al [51].

One shortcoming of CA models are that they poorly model the time taken to traverse space, as the distance covered by agents is limited by the angles allowed. For example, in a square grid world with diagonal movement, the agents can actually travel more quickly if they do so at 45 degree angles, as these moves cover $\sqrt{2i}$ where i is the distance between ajacent cells. If diagonal movement is not allowed, they are forced to perform Manhattan-style movement which overestimates the time taken). By using hexagonal cells instead of squares, this can be mitigated somewhat, though it remains an issue of scale. A further issue, when modelling congestion, is that densely packed crowds of agents cannot move as a unit, but the agents at the leading edge can move forwards into the free space, with the agents behind waiting until the next world update until they have a free space to enter (with the free space propagating backwards through the group. CA also scales poorly with world size; as it requires a cell tiling that covers all of the simulation world (though hybrid simulation such as Chooramun [28] can alleviate this somewhat by replacing less significant areas of the world with graph models).

As CA splits the world into a fine grid of cells, a common approach to path planning is to layer the grid with a "potential field"; this field labels cells with values that represent their distance from exits, or between landmarks that should be visited. The potential field acts as a gradient, with agents moving "downhill" from their current position to adjacent cells with lower values, until they reach the exits. Examples of the potential fields approach to CA directed movement include Kneidl [48], Georgoudas et al [52], Zhang et al [53]. The major limitation of CA with potential fields is that planning occurs at the global level as all agents make use of the same field, and it does not allow for individual agent planning.

2.5.2 Agent behaviour and cognition

Modelling occupant response beyond basic flocking has been investigated in recent years, one such effort on providing emotional response capability to agents is detailed in Sharma's PhD thesis [43] which presents a behaviour model for evacuation scenarios incorporating fuzzy logic techniques to regulate agent emotional behaviour (stress, panic). The author proposes that the implemented multi-agent simulation (with grouping behaviour and collision avoidance) incorporating this emotional behaviour control mechanism is a useful tool for evaluating building layouts and structures for evacuation scenarios. The thesis concludes that room exit RFID scanning is a suitable mechanism for tracking building egress, and validates the implementation of these simulation techniques by comparing with a record of a fire drill performed at Wayne State University.

Pelechano et al's [54] [55] reviews of crowd simulation models and evacuation simulation tools note the limitations of these simulations and emphasising the importance of simulating human psychological/physiological characteristics. They note that individual evacuation behaviour may depend on their knowledge of the building layout, and individuals tend to evacuate via familiar exits rather than via emergency exits which are not generally used as exits in non-emergency scenarios. They expand on the established flocking models by introducing "leader" agents [56], using their simulation to evaluate the impact evacuation leaders and inter-agent communication might have on evacuation times.

Korhonen et al [57] also incorporated individual psychological and physiological behaviours into their Fire Dynamics Simulator evacuation module (FDS-Evac, [58]), introducing modelling of agent panic behaviour and the interaction between agent simulation and fire simulation. Grouping behaviour is also incorporated, with agents forming crowds which move together along the same path to escape. They present the results of a series of experiments to discover the average flow (throughput of agents) through spaces in the building (doors, corridors) when the social parameters of agents are changed (such as changing how eager agents are to push the agent in front of them, how much personal space they attempt to maintain and so on). The authors conclude that a specific set of parameter values (default values) provided a close correspondence to standard evacuation calculations in fire safety guidelines, but also note that their model should be expanded to include more realistic behaviours such as herding behaviour (the tendency for agents to follow each other). Heliovara et al [59] explored the different personalities in evacuation, characterising occupants as patient or impatient, with behaviour governed by a game theory model and implemented through the FDS-Evac [58] simulator. The crowd contraflow behaviour of the FDS-Evac model was validated [60] through comparison with real-world data using a similar approach to what we use in Chapter 3 in this work.

Niederberger [61] presented a Doctoral thesis on multi-agent simulation for video games, describing how to improve a game characters behaviour by implementing decision algorithms in the simulation. The author describes existing models for such simulations notes that game character behaviour in games is generally heavily scripted and lack dynamic response to events. The aim of the presented research was to improve on these models: the appearance of agents in the game should be coherent and remain appealing and hence computation time should be low and distributed such that multiple agents behave in a realistic and appealing manner without impeding the performance of the game. Agents are modelled as a composite of a number of low-complexity components (governing path planning, movement, reactive and proactive behaviour routines, sensory processing etc.) The author presents the agent models as an improvement on prior models as they have support for more sophisticated behaviour but without significant additional computational complexity.

Relation to this work

A huge variety of microscopic pedestrian simulation models exist, many of which make use of flocking and social forces approaches to pedestrian modelling to achieve realistic movement simulation. The EvacSim simulator used in this work makes use of these approaches but by using a reduced complexity social forces approach the complexity of the motion model is kept low with a much reduced set of parameters compared to alternatives, allowing for high-speed simulation without sacrificing motion fidelity. The validity of this reduced complexity approach is demonstrated in Chapter 3, where we compare the EvacSim motion model with real world pedestrian movement observation experiments.

Many multi-agent microscopic models have been developed that incorporate rich and substantial individual behaviours into agents, a particular strength of multi-agent modelling as it enables the simulation of heterogeneous groups of agents in a common simulation world. Development of psychological and behavioural models in agents is an interesting and evolving field, though the difficulty of validating agent cognitive behaviour with real world data limits the current application of these approaches for the real-time simulation for planning support and response for which EvacSim was developed. In Chapter 6 we detail the decomposition of simulation space into multiple future states based on agent groups sharing evacuation route information, more complex information sharing could be incorporated into this approach to simulate branching agent decision states in future work.

2.6 Hierarchical Microscopic and Macroscopic Models

Micro- and Macroscopic models both have their strengths. In particular, macroscopic models are highly scalable, while microscopic models achieve finer detail. In some applications,multiple levels of simulation types have been combined within a larger "hierarchical" simulation model. In these cases, higher detail models can be used for simulation of parts of the simulation world where high detail is required, while the lower-detail abstract macroscopic models can be used for areas which are either simple in both model types (such as a straight roadway) or are not as relevant for the simulation (such as space outside of a viewing area or currently unoccupied parts of a building).

Nagel [62] describes a multi-layer building graph where differing navigation and localization restrictions lead to different graph features; for example, a Wheelchair layer of the graph lacks traversable edges on stairways, while the pedestrian layer allows for edges in those areas. Occupants at a given position in the building are simultaneously present in a room on the topology graph as well as within coverage nodes in the various sensor graphs.

Navarro, Corruble et al [63] [64] use dynamic Level of Detail (LOD) for simulating crowds. The LOD is based on crowd aggregation: agents are aggregated based on the combination of physical distance and psychological distance, grouping agents based on common goals, emotional states and location. Combining these produces an aggregate utility which forms the basis of an aggregation graph. All edges above a certain length are removed leaving several subgraphs, the members of which are aggregated into crowds. The dynamic aggregation approach is similar to the Group Agent approach we consider in preliminary form in Chapter 7 (and is similar to the γ-agent approach proposed by Olfati-Saber [47] , with expanded grouping criteria to incorporate more than just distance as a grouping measure). By aggregating individuals into groups, the complexity of simulation space is much lower and in this work they found substantial simulation time gains with limited loss of accuracy relative to using un-aggregated agents.

Paris [65] et al describe a hybrid collision avoidance model for pedestrian movement combining Fuzzy Steering and Geometric Steering. The application is a 3D Virtual Dublin simulation and the goal is realistic appearance and scalability to achieve large crowds. Using a fuzzy logic selection mechanism, a Level of Detail is selected that balances appearance with complexity. Agents plan routes on a graph derived from Delaunay's Triangulation of the space in Dublin streets, which produces jagged back-and-forth routes which are then converted into more realistic shortest-route paths. This work uses three detail levels:

LOD choice is based on distance and angle from camera with fuzzy logic used to

determine the appropriate LOD; the goal is a realistic appearance rather than realistic simulation. Performance evaluation experiments showed real time-capable simulation for up to 4000 agents, provided that only several hundred agents make use of the high detail level

Kneidl et al [48] [66] introduced an approach to modelling in which two different simulation models are "coupled", an approach that is also used in this work in Chapter 5. Using a microscopic Cellular Automata model, the parameters of a macroscopic flow graph-based model are established to match the results of agent movement in the microscopic simulation. The graph model is derived from the CA model by generating nodes at cells near convex corners of obstacles and connecting these to each other using edges. The capacity of graph edges and the traversal time for these edges is discovered by determining how many CA agents could traverse the space represented by those edges, which is achieved through a control loop which generates occupants at the start of the path in the CA model and instructs them to travel the route. The rate of successful traversal of the route is noted and the loop is repeated with a new introduction rate (higher or lower depending on the achieved results). After several passes through this control loop, the rate of introduction and traversal stabilize and the traversal time and flow rates are established as graph parameters for the given route, reflecting in the Macroscopic model the movement characteristics of agents in the Microscopic model. Our work in Chapter 5 adopts this approach but using a more detailed multi-agent free space social forces model as the microscopic model, and using graphs derived from building geometry and graph simplification methods (described in Chapter 4).

Sung et al [67] approached Hierarchical modelling as a method to improve the performance of a city-based vehicular traffic simulation. Simulation complexity is reduced by modelling the area within a field of view using a high-detailed model of vehicular trafffic movement within the view and using a macroscopic graph approach for the simulation area outside of the view. Cell-and-portal division is used to determine visible scenes and vehicle traffic within this visible scene is modelled using a microscopic vehicle agent model. The area outside scene uses a graph event model to determine travel times for vehicles outside of the scene (with units of traffic traversing the graph until their re-entry) so they re-enter the accurate model at the appropriate time and place (i.e. when returning to user view area). By adopting this approach, the authors show computation time savings relative to using microscopic models for the entirety of the simulation world, and show that use of the event-driven macroscopic model for areas outside of the visible scene does not impact on accuracy.

Chooramun [28] described an approach to pedestrian evacuation modelling in which different parts of the building make use of different simulation methods; by dividing

space into coarse graphs, sparse graphs (nodes represent rooms and are connected to adjacted room nodes with edges) and free space microscopic models. By using simpler models for sparsely populated areas of the building, and more detailed microscopic models for the more significant areas (such as junctions or near exits), the complexity of the simulation world can be adjusted to achieve greater detail where it is most of use, while sacrificing detail in areas where the detail is unnecessary.

Anh et al [68] describe scalable simulation of pedestrian movement on road networks by combining microscopic- and macroscopic simulation detail. In the macroscopic portion of the simulation, the road network is modelled as a directed graph. In the microscopic portion, pedestrians are modelled as agents in free space simulation. The hybrid model uses the microscopic model at junctions of edges and macroscopic model for the middle of the edges. This approach allows for a simple model in areas where the pedestrian movement is orderly and predictable, with more computation effort dedicated to the key area of road intersection where crowd interactions are most likely. Transitioning from one model to the other is based on dematerialization/rematerialization of agents (converting micro agent into flow unit for the graph and vice versa).

Relation to this work
In these hierarchical modelling approaches, the detail of different elements of the simulation space are dynamically adjusted to allow for lower-detail simulation of less significant areas of the simulation world. These approaches achieve reduced computational complexity for microscopic simulation by dynamically adjusting the fidelity of simulation, or use macroscopic models as interrim replacements for microscopic simulation in areas where the simulation is more predictable, less interesting or not currently in view. In Chapter 5 we adopt Kneidl's [48] [66] micro-macro coupling but using higher fidelity, multi-agent social forces simulation rather than CA modelling. With the graph coupled to multi-agent simulation, the two model types are a partnership rather than a hierarchy. The macroscopic model makes use of the microscopic model to provide realistic flow parameters to improve the models accuracy, while the microscopic model exploits the reasoning powers of the macroscopic model to achieve scalable simulation of future states.

2.7 Pedestrian Movement Data

To achieve realistic pedestrian simulation we need to consider the movement behaviour of real pedestrians, and the quality of pedestrian models needs to be validated against real world pedestrian data to ensure its accuracy. A variety of real-world experiments have been performed over the years investigating the movement characteristics of pedestrians in a variety of scenarios, generally characterising move-

ment in terms of a "fundamental graph" (plotting the relationship between crowd density and rate of movement, or "flow"), or in terms of the rate of traversal through bottlenecks of various widths. Due to the logistical and ethical difficulty involved in collection of data from genuine emergencies, the availability of data describing the behaviour of pedestrians in various real-world emergencies is low, though some examples exist through video analysis [69] and post-emergency questionnaire survey [70].

Validation of pedestrian simulation models in this work (Chapter 3) is performed through comparison with real-world experiments involving crowds in bottlenecks and in T-junctions. The bottleneck experiments used for validation are: Seyfried [71], Kretz [72], Muller [73] and Nagai [74]. The movement of crowds through T-junctions is compared against the work of Zhang [1] [2]. The general setup of these real-world experiments is to create a door or corridor with a particular width (e.g. 1.5 metres) and monitor the traversal of pedestrian participants through the space (after which they loop around and begin again.

Data is collected (either by observation, or by algorithmic analysis of video data to extract density and velocity data) to determine the rate of travel possible for the given width. These experiments then vary the width of the door or corridor and this achieves a range of traversal rates ("flow") corresponding to the varying bottleneck widths. Zhang et al's experiments [1] [2] investigated crowd behaviour in a fixed-width T-junction. In these experiments, pedestrians arrive from two branches of a T-junction and merge to form a single stream moving forwards (Figure 3). The data collected in this experiment shows the relationships between crowd density and the velocity at sampling points before and after the merging. This work was used to further validate the EvacSim agent model (Chapter 3) and motivated some adjustments to the agent model in order to achieve similar results.



Figure 3: Zhang [1] T-junction real-world pedestrian experiment

While data on real emergencies in the sort of detail possible in experiments is extremely rare, post-disaster case studies do exist. Fischer's case study [75] on the Ephrata evacuation provides useful insight on the attitudes of individuals in emer-

gencies, investigating why different people behave quite differently in these kinds of events. A similar study was performed by Sekizawa [76], investigating the behaviour of occupants of High-rise apartments in Hiroshima during the apartment fires of 1996. Helbing et al [69] analysed video of the Hajj stampede disaster in 2006, leading to a series of recommendations which have been implemented to modify the infrastructure at Mecca.

Relation to this work
In Chapter 3 we describe in detail the pedestrian movement model used in EvacSim, and validate this model by comparison with a variety of real-world pedestrian movement experiments. The availability of real-world pedestrian movement data is of great benefit in validation of pedestrian simulation models. While most of this real-world data deals with movement through bottlenecks, Zhang et al [1] [2] dealt with the case of crowd merging at junctions, an important consideration for modelling of crowds during evacuation.

2.8 Hazard modelling

The goals in hazard modelling vary by application; some fire simulations can be extremely computationally intensive, requiring distributed supercomputers and long computation timeframes, with detailed material models, and are well-suited to post-event forensic analysis or detailed risk assessment. Other fire and smoke simulation approaches are approximate, computing in close to real-time but require constant adjustment to match up with real observations. We present here some releated work in hazard modelling for contextual purposes, in future work we will consider the integration of more detailed hazard models with the EvacSim simulation platform.

Chu et al [77] presented a dynamic risk analysis approach that maps the risk to occupant in the building based on sensor reading during emergency. By combining oxygen levels, presence of toxic gases and temperature, a map of occupant risk can be produced for the building.

Himoto [78] and Bukowski [79] model fire spread as the transference of heat from areas on fire to adjacent areas (such as rooms or nearby buildings), modelling in detail the physical transference of heat, fire flumes and radiated heat in highly complex simulations incorporating physical material properties.

The Fire Dynamics Simulator (FDS) + Smokeview [80] is a Computational Fluid Dynamics-based fire simulation model which simulates fire and smoke spread as fluid dynamic problems. FDS can take converted CAD drawings of buildings as input and simulate the slow of fire and smoke in building spaces.

Koo et al [81] developed a fire modelling approach which dynamically adjust model

parameters in response to sensor reading updates during emergency. While this model is less detailed than complex fire simulations such as Computational Fluid Dynamic approaches, it can be computed in real time during emergency and used as a predictive support tool. By reacting to real sensor readings, the model can be adjusted to match more closely with the actual fire spread and over time the simulation accuracy is improved through these adjustments. This approach to hazard simulation is good fit for the realtime simulation decision support approach as it allow for hazard spread to be modelled in step with the actual emergency and provide some measure of near-future high-level prediction, which while less accurate, responds to the evolution of the emergency state.

2.9 Conclusions

Simulation and planning in emergencies is an evolving field, the explosion of simulation models in recent years reflects the variety of competing requirements for different applications in evacuation modelling and the different strengths of micro- and macroscopic models. Simulations making use of complex social force, multi-agent models have realistic pedestrian motion. While computationally intensive, existing work has achieved performance gains by organising simulations hierarchically and reducing the fidelity of less critical parts of simulation environments or aggregating agent behaviour at the group level.

Coupling of microscopic simulation with macroscopic flow graph models allows for automated configuration of flow parameters, improving the realism of flow models without increasing their complexity. Coupled flow graphs can be used by dynamic evacuation planners to compute evacuation plans based on the dynamic state of a building in emergency, and by using microscopic simulation we can evaluate the performance of planners in simulated emergencies with a variety of plan allocation strategies and identify issues that would not be apparent through macroscopic simulation.

A further opportunity is the use of macroscopic flow graphs for simulation decomposition. Existing approaches to simulation scalability decompose the problem geometrically, separating occupants based on their physical location and simulating them separately. By analysing the flow graph of buildings, we can separate the occupants based on both their physical location, *and* their future positions, based on their estimated traversal times through spaces in the building.

3 Details and Validation of the Microscopic Agent Movement Model

3.1 Introduction

Central to the utility of microscopic pedestrian simulation is the quality of the agent movement model. While planning behaviour, sensing and communication have an important bearing on the activity of occupant agents, this is of little use if the fundamental motion of agents is unrealistic. In this chapter we expand on the features of the EvacSim pedestrian agent (described in Appendix A.3.2) describing the parameters governing agent motion allowing for agents to move towards arbitrary goals while avoiding collisions with other agents or obstacles. The agent motion model is kept simple to keep the computational complexity low but features the basic requirements for agent motion. This simple model is then validated against real-world pedestrian movement data to demonstrate that this simple motion model provides a high level of realism.

To validate the agent model, we perform experiments to evaluate the model in terms of the pedestrian throughput in various bottleneck widths and compare these results with real-world pedestrian experiments from the literature, demonstrating a close correspondence between EvacSim pedestrian movement and real behaviour. We also investigate the behaviour of crowds merging in building evacuation, comparing EvacSim's model with real-world T-junction results and we show how modification to the braking behaviour of the agents improves the model realism.

3.2 Requirements

Simulation of evacuation requires an occupant model that accurately reflects the behaviour and movement of building occupants in order to properly reproduce key evacuation metrics such as congestion data and evacuation time. Agent motion is governed by a combination of personal space preservation, obstacle and collision avoidance (as described by Helbing [41]), and should reproduce the queuing and congestion phenomena which limit pedestrian traversal of spaces. Realistic agent motion and use of space is a requirement for building and planner evaluation, and is central to the utility of micro-macro flow coupling described in Chapter 5.

In application scenarios involving real-time response systems such as dynamic evacuation planning, there is a strong requirement for efficient, faster-than-realtime simulation to produce predictive results to assist decision making. This requirement necessitates a balance of realism and computational simplicity in order to provide useful simulation results under the time constraints. In multi-agent simulation, the

overall complexity of the simulation is a product of the individual complexity of constituent agents. To achieve a balance of speed and realism, complexity is kept low by performing relatively simple atomic actions repeatedly over time. For example, rather than re-computing agent positions to maximize agent personal space in each tick of simulation time, our agents make a series of small positional adjustments over the course of multiple ticks, with stable positioning arising naturally as a consequence.

3.3 Model Details

In Appendix A we describe how EvacSim models the world in two-dimensional free space, representing distance in simulation "units". Time is represented as simulation "ticks"; in each tick of the simulation, each simulation element (occupant agents, sensors etc) is updated according to its function or behaviour. Agents are represented in the world by 10-unit diameter discs and possess a motion vector combined with a two-dimensional (x,y) floating point coordinate pair which represents its position in space and is manipulated by the agent to govern moment-to-moment movement.

The maximum amount of forward movement a given agent can make is taken from a standard distribution of maximum speeds, randomly selected on creation of the agent. In this way, a population of agents features a variety of possible maximum speeds, though in practice their rate of forward movement performed tick-to-tick is governed by this value combined with the braking procedures of the movement model. Agent decision making processes modify this vector periodically, directing it towards a goal or adjusting it to avoid collisions depending on the agent high-level behaviour model and agent observations. Agents select goals in their line-of-sight to move towards as part of their decision-making behaviour by reasoning about a graph model of the building space. In Chapter 4 we describe the generation of building navigation graphs which the agents can use to navigate large spaces by treating paths on the graph as sequences of sub-goal Coordinates.

The agents attempt to preserve personal space around them, and avoid collisions with other agents by braking and steering to avoid them, similar to the social forces model described by [41]. Each agent possesses a number of attributes (Table 1) governing the parameters of the movement behaviour, and a set of global constants dictate the parameters of general simulation, such as the amount of personal space agents will seek, or the distance in front of agents where collision avoidance is considered. In the validation work in this thesis, 20 simulation ticks corresponds to 1 second of simulated time, and 15 units corresponds to 1 metres of space. With this conversion rate, agent maximum speeds fall into the range of 1.5-2.5m/s. These

Table 1: EvacSim agent variables

Variable Name	**Description**
PS	Amount of personal space preserved around each agent if possible (in simulation units)
σ_i	Factor determining the rate of personal space adjustment agent i makes from tick-to-tick (in simulation units)
υ_i	The maximum distance agent i can traverse in a single tick (in simulation units)
$\upsilon_i^{'}$	The distance the agent i will traverse in the next tick, determined by $\upsilon_i * \beta_i$ (in simulation units)
θ_i	Facing direction angle for agent i in degrees
pos_i	two-dimensional floating point coordinate pair representing agent i's position in space
$goal_i$	two-dimensional floating point coordinate pair representing the agent i's current goal
$GOAL_{adj}$	Amount of adjustment permitted to SteeringVector angle each tick when pointing towards goals, in degrees
OD	Distance in front of the agent to check for obstacles, in simulation units
OBS_{adj}	Amount of evasive action taken per-tick when avoiding obstacles, in degrees
β_i	Floating point value in [0,1], Used with υ_i to determine $\upsilon_i^{'}$ value
ω_i	The radius of the agent i (in simulation units).
CD	Distance in front of the agent to check for collision with other agents (in simulation units)
PS	Distance around agent to check of maintaining personal space (in simulation units)
ϵ	Angle cone in front of agents in which upcoming collisions are considered (in degrees)

conversion rates were chosen to produce smooth animation of real-time simulation and (using a 1-unit to 1-pixel translation) display of a large area of simulation space within a single display screen in native resolution without image rescaling.

3.3.1 Agent Tick procedure

Each agent is called upon, once per simulation tick, to update its position based on personal space, collision avoidance and progress towards its goal. This procedure operates as follows:

1. Observe area for obstacles and other agents (line of sight checks, cf. Section A.2.3)
2. Adjust Personal Space (Section 3.3.2)
3. Direct θ_i towards $goal_i$
4. Adjust θ_i to steer around obstacles
5. Adjust pos_i to sidestep obstacles
6. Apply braking to avoid collisions with other agents (Section 3.4.1)
7. Move forward v_i' units, in θ_i direction

3.3.2 Maintaining Personal Space

Accuracy of the agent model requires a mechanism for the agents to ensure that they are not overlapping each other. Typical behaviour for humans is to avoid close contact with other occupants if there is space to do so, maintaining a gap between other occupants, or personal space.

We achieve this by performing checks (once per tick per agent) to determine if there are any agents nearby that are within a threshold Personal Space distance (PS) and for the agent to adjust its position to move away from the closest infringing occupant (movement amount based on σ_i). Through this mechanism of repeated observation and position adjustment, a compressed group of agents should spread out to cover an area. An agent is considered to be violating another's personal space if the distance between the centres of the two agents is less than the sum of the radii) plus PS. Having identified the closest other agent that is within the PS threshold, the agent moves in the opposite direction by an amount based on the agent Space Adjustment value σ_i and the distance between the two agents i, j according to Formula 1).

$$\left(\frac{PS}{distance(pos_i, pos_j)}\right)^2 * \sigma_i \tag{1}$$

This value indicates how far the agent pushes away from the nearest occupant in a single time unit, and scales the amount of movement based on inverse square of the distance. This personal space behaviour is implemented as part of the agent "tick" method, called once per time unit. After all agents have moved to accommodate their personal space requirements, the simulation ticks forwards and the procedure repeats. In these scenarios, the agents will move apart again in the next few ticks if possible but for very dense populations of agents there may be no stable positioning without personal space violations occurring.

3.3.3 Steering towards Goals

A crucial capability agents need is the ability to move from their current position towards a goal. This fundamental capability can be combined with path planning to produce a variety of microscopic movement behaviour. The basic steering mechanism is based on the agent pointing themselves towards their current goal, and adjusting their position from time unit to time unit, to move themselves closer to the goal. This steering of an agent i is performed using a Steering Vector angle θ_i and the movement amount is governed by υ_i'. The rate of steering adjustment is given by $goal_{adj}$; this value governs the maximum adjustment (in degrees) that an agent can perform within a single tick.

3.4 Avoiding Obstacles

With the Steering Vector angle, the agent can direct itself towards a goal and move towards it. Without obstacle avoidance, the agent would bump into obstacles or other agents close to its chosen path. By adjusting the Steering Vector to steer around obstacles in its path, the agent can safely travel to its goal. To adjust the Steering Vector, the agent first needs to look ahead to determine if it is approaching an obstacle. To do this, the agent creates two "Eye" vectors. These vectors are projected out directly from the sides of the agent by an amount equal to the Obstacle Lookahead distance, OD, and aim in the same direction as the Steering Vector θ_i.

The lines described by these Eye Vectors are checked against the building geometry to determine if there is a collision coming up (Figure 4). If there is a Collision on the "Left Eye" but not on the "Right Eye", then the upcoming obstacle can be avoided by steering more to the right; similarly if there is a collision on the Right Eye and

not the Left the agent steers left to avoid it. The agent then also adjusts its current position by "sidestepping". This movement causes the agent to step 1 unit to the left or right (i.e. perpendicular to the steering vector) if there is a collision on the Right or Left Eye. If both Eyes are in collisions, then the agent slows down to avoid impacting the obstacle while the steering vector is adjusted to steer away over the next few ticks. The amount of collision avoidance steering adjustment is governed by the parameter OBS_{adj}.

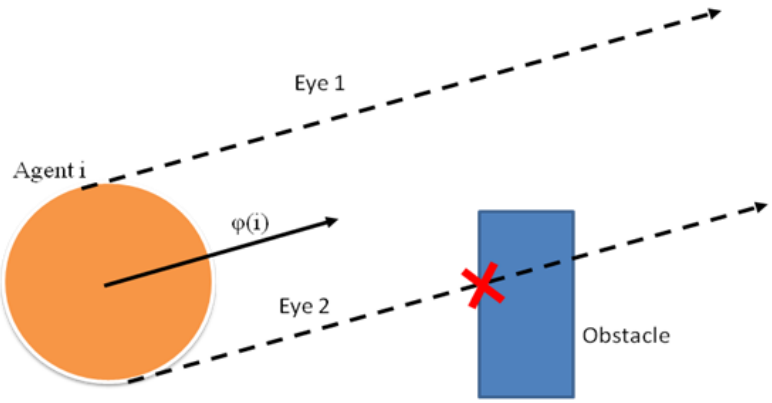


Figure 4: Agent Lookahead for Avoiding Obstacles

3.4.1 Braking and Avoiding collisions between Agents

If an agent is approaching another agent, it needs to slow down to avoid a collision. To produce behaviour such as queuing at bottlenecks, the agents need to be able to slow themselves down to avoid collisions and to stay at this low speed while waiting for the area ahead to clear up. This braking behaviour is achieved by periodically scanning the area ahead of the agent for other agents in line of sight within a vision cone delineated by ϵ (Figure 5). The agent then adjusts the impact the υ_i has on movement by multiplying it by a brake factor value β_i, between 0 and 1.0. β_i is based on the proximity of the nearest other agent $_j$ and the angle difference between θ_i and θ_j. This is implemented by iterating through all other agents in the agent's purview (within its Zone and Halo Region: Appendix A.4.4), and placing any agents within a set number of degrees (Brake Angle parameter ϵ) to the left or right of the Steering Vector in a "Brake Candidate" set. The distance to the nearest other agent in this set determines the amount of braking to be applied to υ_i (through multiplication) to produce the final movement amount, υ_i'.

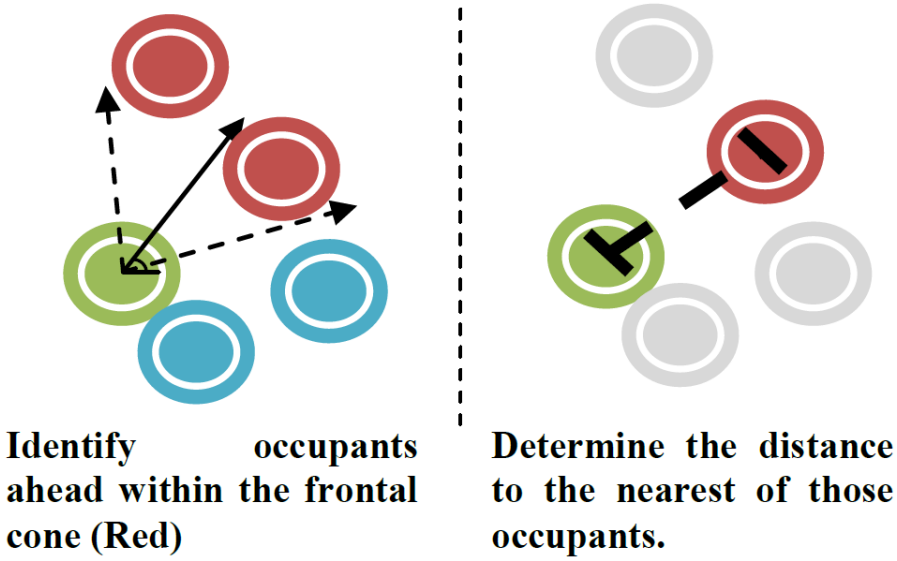


Figure 5: Agent braking to avoid collision

3.4.2 Resolving Overlap ("Enforce core")

In some cases, agents may be in a collision with each other (i.e. their radii overlap). This can occur with large populations of agents within confined areas where no amount of adjustment can produce a configuration without collisions, or when fast moving agents come into contact with agents that are moving much more slowly and have not sufficiently slowed down yet (as braking takes several ticks to slow the agent down completely). In such a situation, the agents should prioritize the resolution of the overlap over their goal-directed movement. To achieve this, we implement an "Enforce Core" behaviour invoked during collisions which disables goal-oriented movement until the overlap is repaired (via Personal Space adjustments described in 3.3.2). This behaviour is so termed as it enforces that the central core of the agent is free from overlapping agents before other movement is performed (though the wider personal space might still be impeded).

Implementation of the Enforce Core behaviour is tested by generating 400 agents in a room on the left-hand side of the simulation world and sending them through a narrow doorway to the right. The number of individual instances of agents in an overlapping state is recorded each tick and averaged over the course of 600 ticks.

The over-supply of agents causes high congestion at the bottleneck area, and without the Enforce Core mechanism (Figure 6) we find a high degree of agent overlapping as the pressure of agents trying to get to the goal exceeds the force of the agents attempting to maintain personal space. This high degree of overlapping results in a very large compression of agents within an unrealistic space; the number of occupants that can be packed into a space is excessive as they are physically occupying almost the same positions as each other.

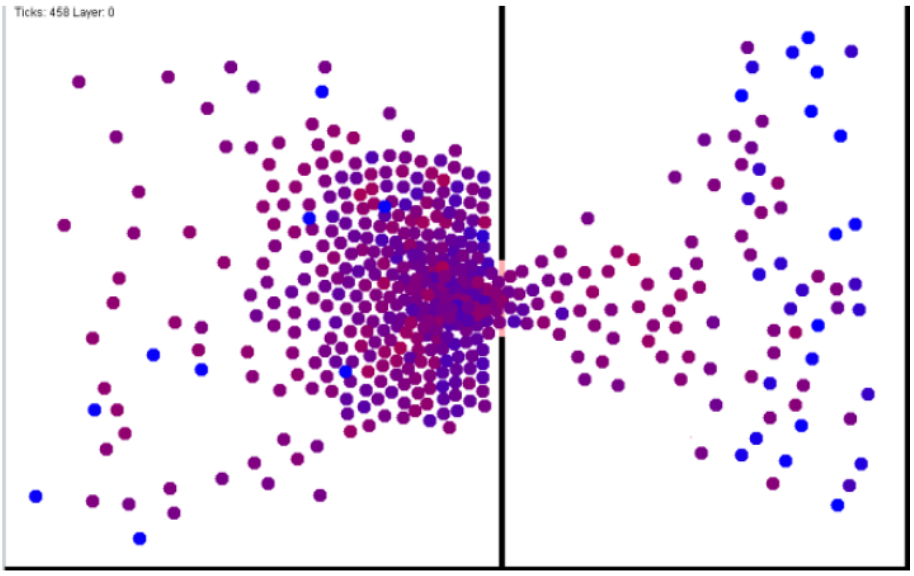


Figure 6: Dense crowds without core enforcement.

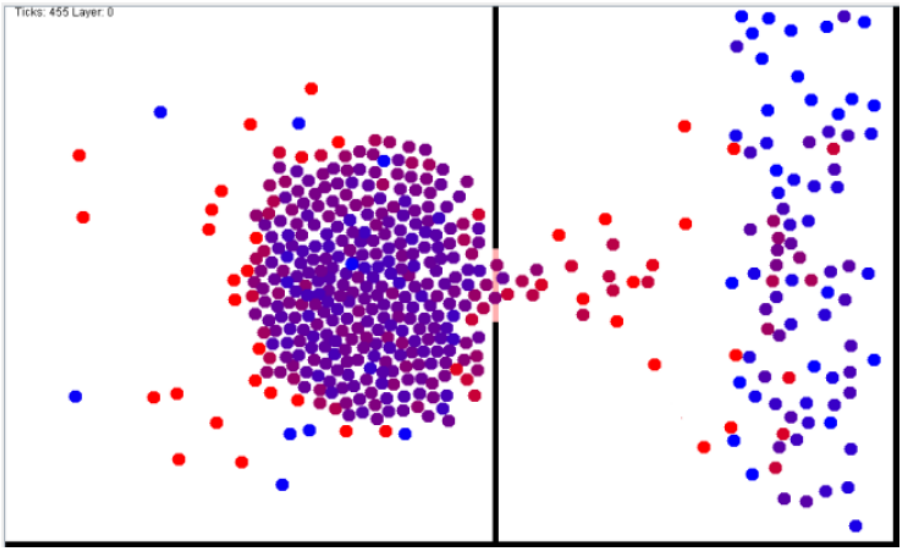


Figure 7: Enforcing Agent core in dense crowds

With Enforce Core enabled (Figure 7) we find that the goal-movement pressure is reduced as agents stop to correct for personal space whenever overlaps occur, and there is a much more stable configuration near the doorway. Using this approach leads to an average number of overlapping events per tick of ∼0.18 over the course of the experiment compared to an average of ∼5.4 agents overlapping per tick when Enforce Core is not enabled. Agents are compressible but only rarely when using Enforce Core, and this corresponds with the assertation of Hoogendorn [82] that compression should be allowed but should also be avoided when possible.

3.5 Validation Experiments (Bottleneck)

The EvacSim pedestrian model was tested in a series of experiments to compare with real-world pedestrian movement experiments. Bottleneck throughput is a significant metric for evaluating egress simulation models, as the number of occupants that can be safely moved through a given area of the building is a key element of evacuation planning. For adequate decision support, evacuation simulation needs to produce throughput results in line with real-world pedestrian behaviour.

The first set of experiments was performed to determine the flow rate of pedestrians through a bottleneck. This experiment involved generating agents and instructing them to walk through a narrow corridor and out the other side, where they are removed from the simulation. The experiment is repeated with a variety of bottleneck widths and the maximum flow throughput is recorded in each case.

First we set the experiment up with the widest corridor to be tested (2m). Agents are introduced in the left-side room and instructed to move to the right. The first introduction rate tested is initially set low (1.25/second), with the procedure repeated for progressively higher introduction rates (up to 6.66/second); and the actual throughput achieved is noted for each inflow rate attempted. The purpose of this procedure is to discover an optimal throughput value for the corridor. At low supply rates the throughput achieved increases linearly but at higher supply, we observe lower throughput (Figure 8) caused by crowding at the entrance and inter-agent jostling, corresponding with the traffic model results of Nagel and Scheckenberg [83].

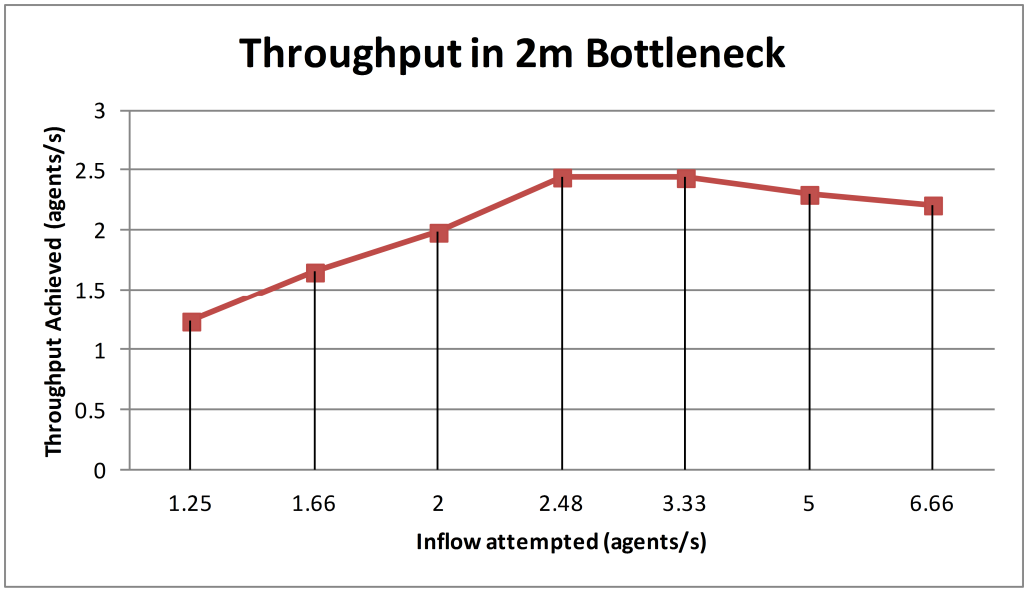


Figure 8: Throughput Achieved for 2m corridor

The experiment is then repeated varying the corridor width, and gradually increasing the agent supply until the throughput peaks, giving the optimal throughput for each corridor. These results are compared against the real-world data described in

Section 2.7 in the graph of Bottleneck Throughput (Figure 9). In this comparison we observe that EvacSim throughput results fall well within the range of these real-world results. These results shows that EvacSim's pedestrian model accurately reproduces bottleneck congestion and throughput found in real pedestrian behaviour. We observe an underestimation of throughput for very narrow bottlenecks (70cm and below); however bottlenecks of this size are not likely to be a feature of building evacuation as they would be more narrow than the minimum door width typically stipulated by building regulations [84] [85] (80cm or more).

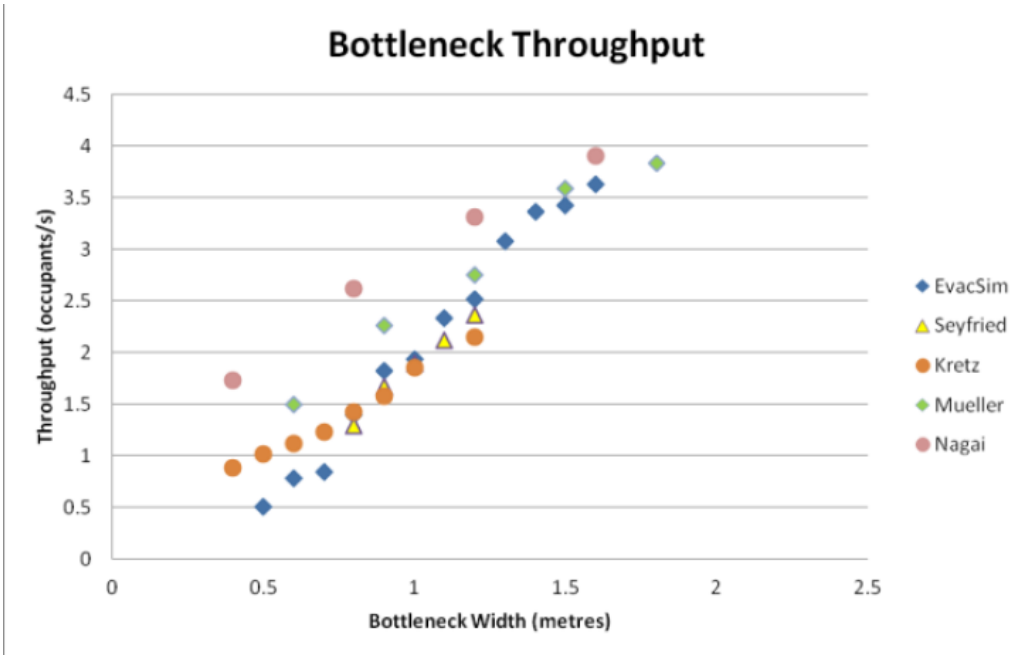


Figure 9: Bottleneck throughput comparison of EvacSim with real-world experiment results

3.6 Validation Experiments (T-junction)

Crowd merging is a key feature of building evacuation, as small groups in the building head towards exits and come together into larger groups. Particularly significant for evacuation planning are the movement characteristics of occupant flows merging together. Zhang et al [1] [2] performed real-world experiments on crowd merging in T-junction spaces and identified a phenomenon wherein the average velocity for a given density of occupants is lower at the point of merging than it is in the space after the merge occurs. They suggest that the cause of this is likely to be slowdown from negotiation of the merging of two streams and tentativeness.

3.6.1 T-junction experiments with default agent

Experiments were performed in a scenario based on the T-junction pedestrian movement experiment performed by Zhang et. al [1] [2] (Figure 10). In these experiments,

pedestrian agents are introduced at two sides of a T-junction (the black and red participants in the Zhang experiment in Figure 10), and instructed to move to the point of junction and then continue down the central route. All corridors have the same width, and a grid of square simulated "sensor" regions is used to periodically sample the velocity and density values in different areas of the T-junction space. J_s represents the average density versus velocity; this is computed through video analysis by generating Voronoi cells as $< \rho >_v . < v >_v [1/m.s]$) versus $< \rho >_v / m^{-2}$ (occupants per m^{-2}) in Zhang. EvacSim J_s values are computed by sampling the number and average velocity of agents present within a grid of 10-unit×10-unit square tracking regions.

The colours of agents in the right-side part (EvacSim) of Figure 10 display their averaged velocity over the last 15 ticks, a range from blue to red representing at rest (blue) to the highest possible speed given from the standard distribution of speeds (red). We label appropriate sensors as "Before", "Centre" and "After"; Before and After correspond to the "in front merging, left" and "behind merging" areas of [1]. "Centre" is the merging area of the T-junction, an area unlabelled in [1] .

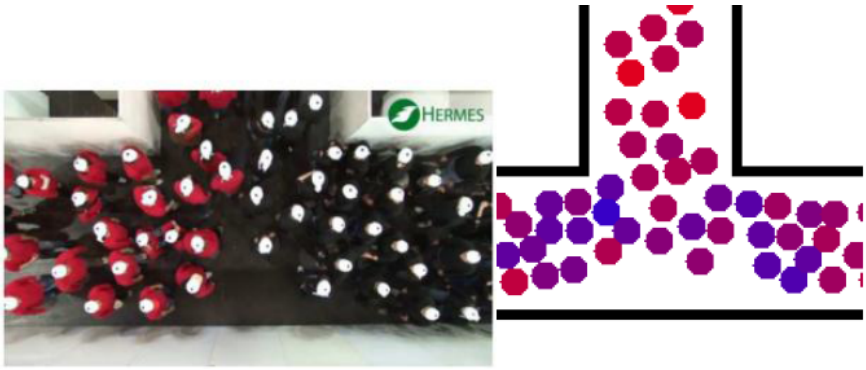


Figure 10: Zhang (left) and EvacSim (right) T-junction experiments

As with Section 3.5, we start with a low inflow of agents and gradually increase this inflow over time. The results of each sensor sampling are given as data points in the Specific Flow graph (Figure 12) in as J_s (average velocity for a given density of occupants. With the standard agent model, we find that the Before, Centre and After results are similar, in contrast with the data from [1], in which "behind merging" shows a clear seperation from the areas before the merge (Figure 11). The standard agent model does not differentiate between agents coming towards one another (i.e. two streams merging) and agents heading in the same direction in a single stream; agents will apply braking even if the agents in front of them are also heading in the same direction.

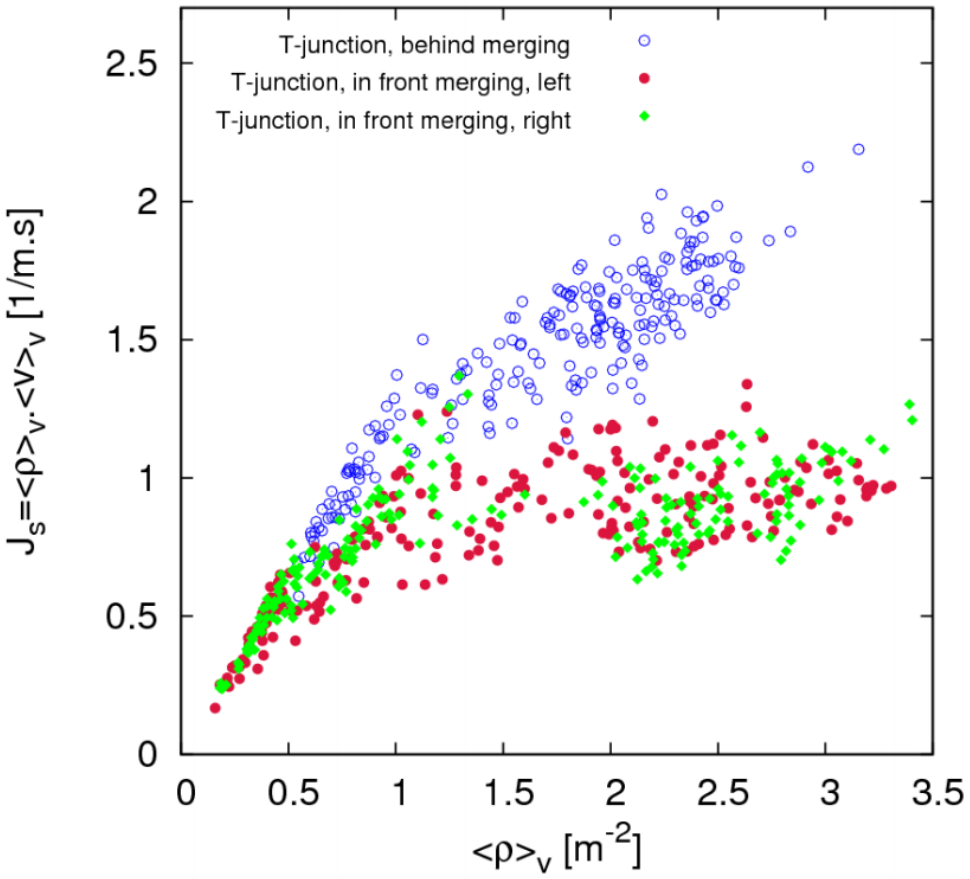


Figure 11: Flow vs Density in real-world T-junction experiments (taken from Zhang [2])

3.7 T-junction experiments with stream formation behaviour

To reproduce the phenomenon identified by [1], the agents need to distinguish between two different cases when another agent enters the braking area delineated by ϵ and CD (Table 1). These cases are whether the agents considered for braking are travelling approximately in the same direction as the agent or not. To achieve this, we extend the braking behaviour to consider the average vector of recent movement for agents in the braking area. If a brake check would normally occur, but the agent in front is heading in the same direction as the braking agent, then the braking agent can maintain its current speed, which we call "stream formation". Agents in a crowd moving in a common direction should travel at a faster average speed due to the lower degree of braking occuring.

Each agent possesses a record of recent positions which is used to produce an average past motion vector by finding the mean direction and distance of movement between each prior position in the record (mean is used for simplification of computation). Agents observe one another over time to produce past-motion vectors for other agents and use these to determine the difference in motion angle between one another, the "motion difference angle". We modify the braking procedure described in Section 3.4.1 to ignore braking for agents that are within the obstacle lookahead

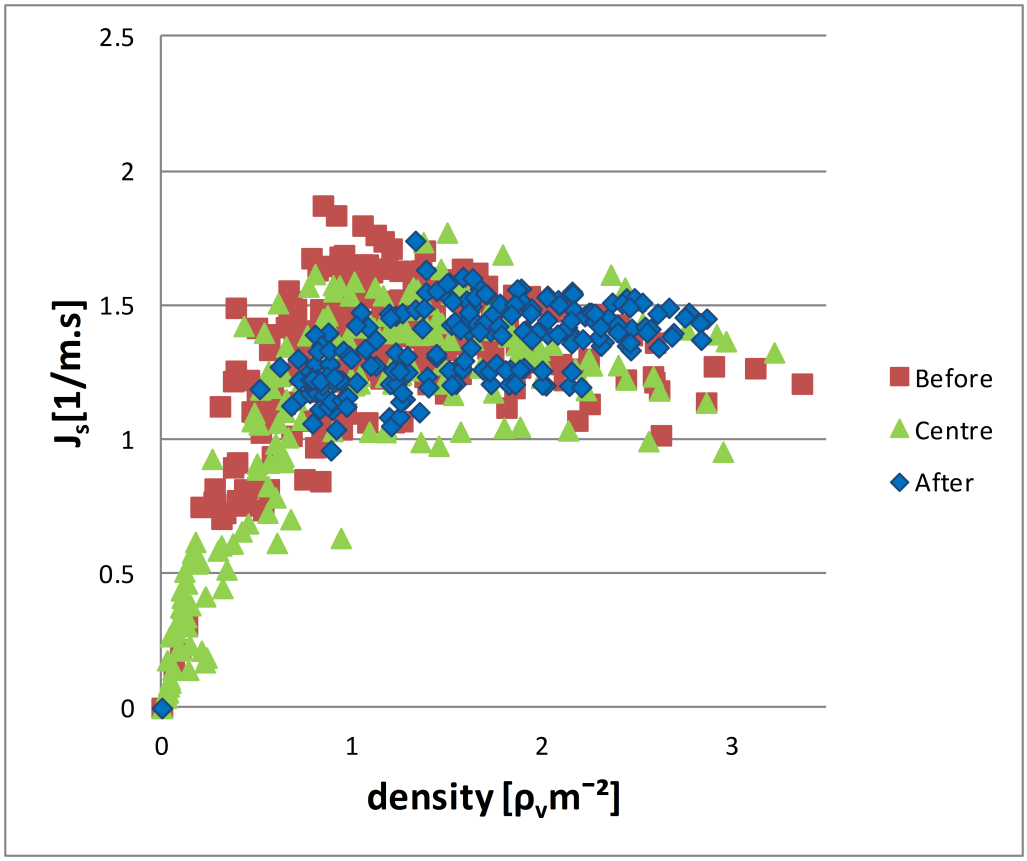
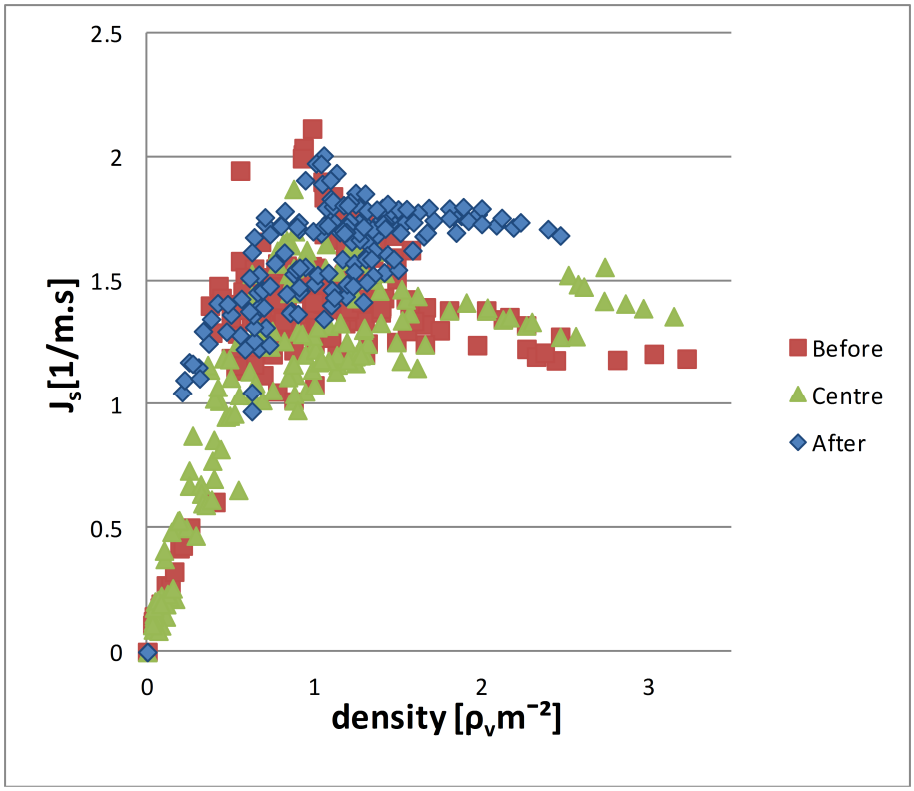


Figure 12: EvacSim T-junction Flow vs Density (default agent model)

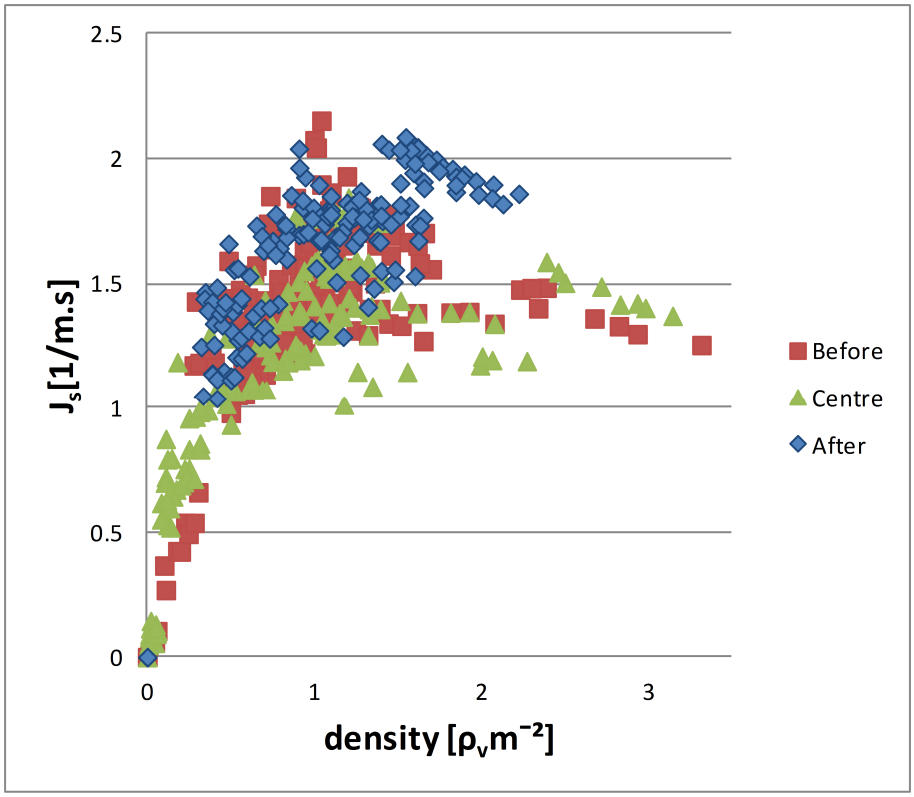
(CD) distance, but have a similar angle of motion (a low "Motion Difference Angle", a difference lower than the parameter "MotionDifferenceAngle tolerance", or "MDA". While braking should still occur when other agents are very near (we use 25% of the CD value as the threshold), this allows for streams of agents travelling in approximately the same direction to travel at a normal speed, and by still enforcing braking at very close distances, we can ensure than a sudden stop by one agent will cause the agents behind it to slow down once they get closer.

The experiment described in Section 3.6.1 was repeated using two MDA values: 20° and 60°, with the relationship between specific flow and density (J_s) shown in Figure 13. These two values were chosen to investigate the impact of the amount of leeway the agents have before considering agents ahead of them to be travelling in a different direction. 20°gives a tight restriction on this tolerance, allowing just enough leeway to allow for the small left/right adjustment agents naturally make. The higher tolerance of 60° gives the other extreme; low-braking streams form in more of the experiment space (appearing in more of the turning area) and the separation between areas where streams form and where the turning and merge negotiation occurs is less distinct as a result.

We find a clear separation between the area after merging and the areas before the merge when using the stream formation behaviour. We also observe greater



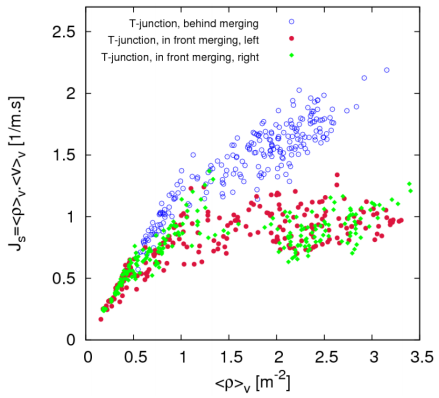
(a) Stream Formation, MDA = 20



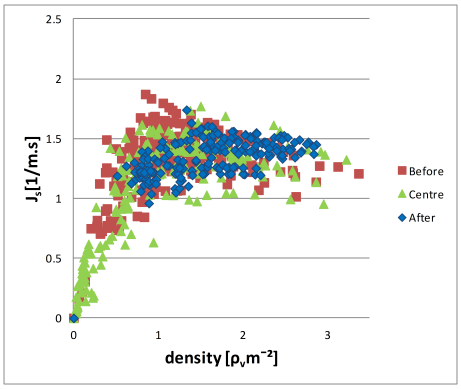
(b) Stream Formation, MDA = 60

Figure 13: T-junction Flow vs Density (Stream-forming agent model)

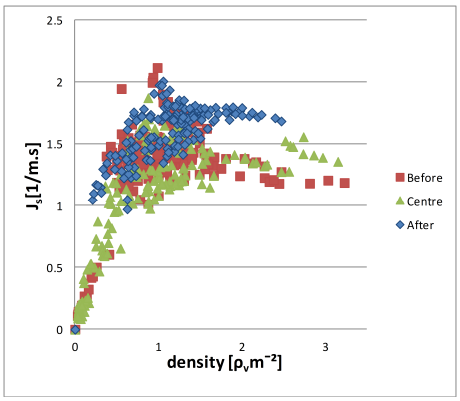
average specific flow due to less braking occurring in the whole system generally. For comparison we show these results with the Zhang [1] results in Figure 14.



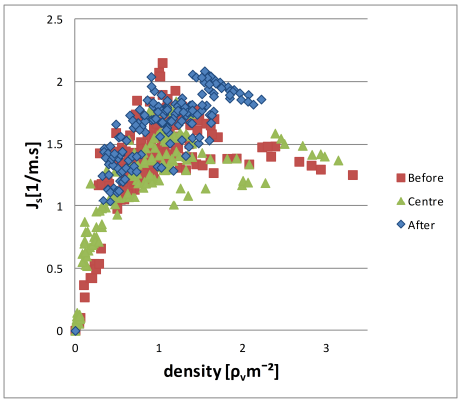
(a) Zhang t-junction



(b) Default agent



(c) Stream Formation, MDA = 20



(d) Stream Formation, MDA = 60

Figure 14: Zhang [1] and EvacSim agent t-junction Flow vs Density comparison

3.7.1 T-junction Heatmaps

Data from the square grid "sensors" is averaged out over the duration of the experiment and used to produce heatmaps for comparison with heatmaps from [1] (Figure 17) illustrating the average densities (Figure 16) and velocities (Figure 15 for agents in the T-junction experiment space (Figure 10).

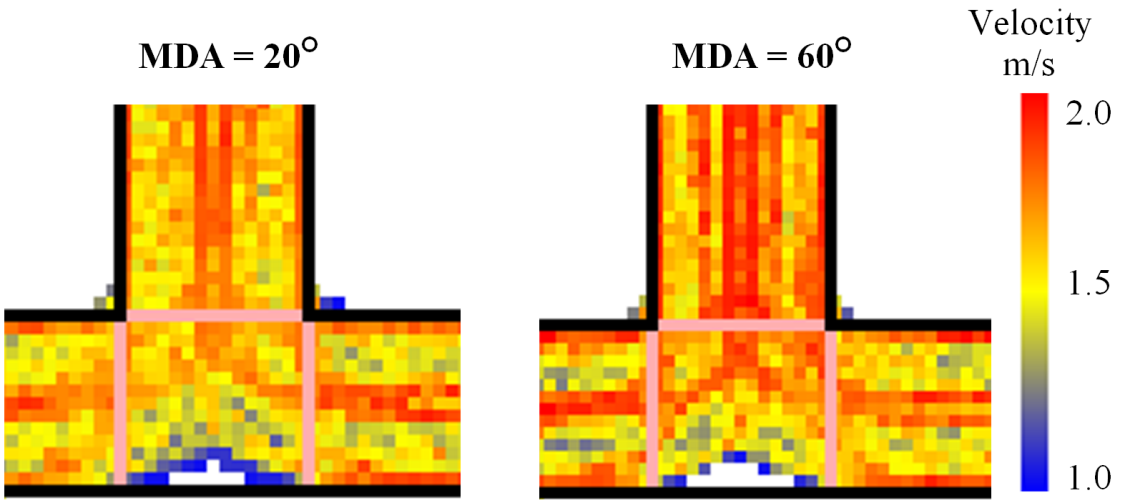


Figure 15: Average Velocity Heatmaps for EvacSim Stream-forming agents, MDA=20, 60

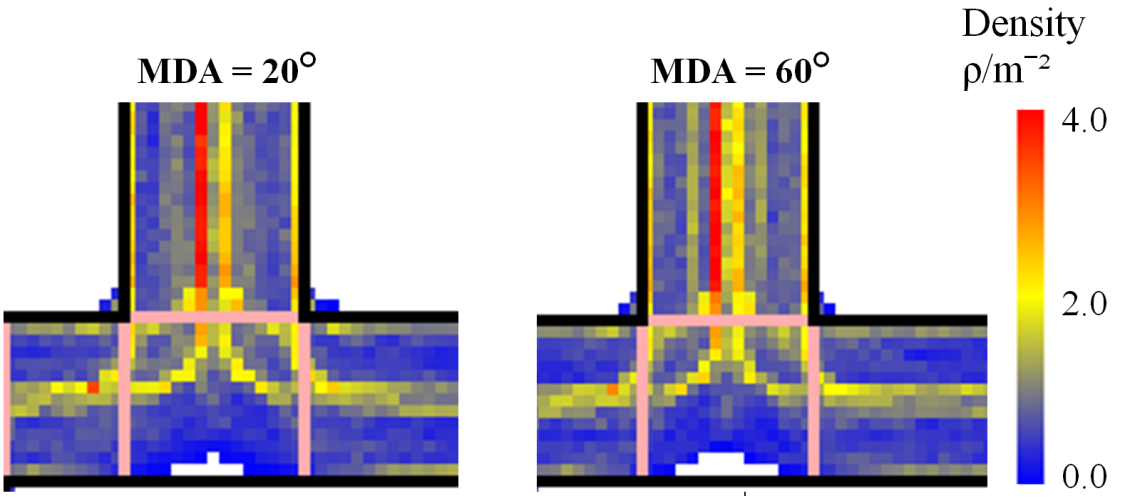


Figure 16: Average Density Heatmaps for EvacSim Stream-forming agents, MDA=20, 60

In these heatmaps we can observe the impact of a wider tolerance in Motion Difference Angle. With MDA of 20° we observe the velocity of agents in the "centre" area to be lower than the mid-corridor flow after merging, as in [1]. With MDA of 60°, the velocities through the merging centre area remain high as the agents are more likely to ignore the braking procedure despite negotiating a merging.

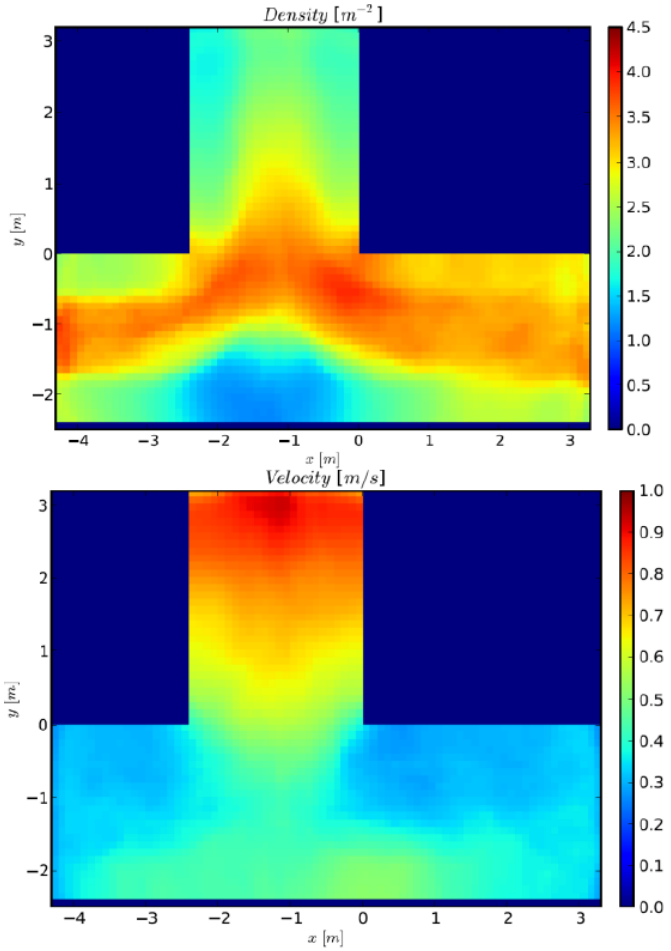


Figure 17: Density and Velocity heatmaps for Zhang [1] T-junction experiments

3.8 Conclusions

In this Chapter we described the requirements and details of the EvacSim pedestrian evacuation model. This model achieves complex occupant behaviour through periodic, low-complexity computation to produce realistic pedestrian movement. The accuracy and realism of this model is of key importance in evaluating emergency evacuation planners and providing useful predictive information for safety personnel. To show the model realism, it was validated against real-world pedestrian movement experimental data in bottleneck throughput scenarios, demonstrating that this evacuation simulation model produces throughput metrics that match closely with real-world experiments investigating pedestrian movement.

We identified the special case of crowd merging in building spaces as being of particular importance in evacuation. We made adjustments to the agent model to produce a seperation phenomenon similar to that identified in real-world experimental results for crowd merging in T-junction spaces. These adjustments reproduces the phenomenon of reduced flow in crowd merging areas relative to unidirectional corridor flow.

With a validated microscopic agent movement model, we can exploit the EvacSim

simulator to establish flow parameters for macroscopic flow graphs, a procedure termed micro- macro- coupling, which we investigate in Chapter 5.

4 Hierarchical Level-of-Detail graphs for Evacuation Simulation and Planning

4.1 Introduction

Simulation and planning of pedestrian evacuation of buildings benefits from the availability of Macroscopic building topology graphs. A variety of graph detail levels is useful for different applications within planning and simulation of evacuation, such as microscopic agent navigation and macroscopic dynamic evacuation planning. In this chapter we investigate Topological Graph generation from building geometry, presenting techniques to generate multiple levels of graph detail in a manner that allows for easy translation between different detail levels for different purposes.

By first generating a "fundamental" graph (derived based on space partitioning as described in Appendix A.2.4, we can then apply graph simplification operations to produce high-level, low-detail graphs suitable for use with a macroscopic evacuation planning algorithm. Similarly, we can increase the detail of the fundamental graph by introducing extra nodes and edges to produce very dense navigation mesh graphs which can be used by agents in microscopic simulation to plan movement routes in a detailed and intelligent manner.

4.2 Graph Detail Levels and Requirements

Microscopic agent-based simulations make use of graphs for agent navigation planning, allowing agents to reason about traversability of space and plan routes between goals. Macroscopic dynamic evacuation planners use occupancy information combined with network flow graphs to compute evacuation routes that minimize evacuation time and hazard exposure. While both of these models make use of graphs, the requirements of each are different. Simulated occupant agents require high-detail graphs to provide the greatest fidelity in movement and planning in the local area. On the other hand, dynamic evacuation planners work best with low-complexity graphs with small numbers of nodes and edges, not only to minimize computation time in time-sensitive evacuation scenarios, but also to model congestion in the graph as occupancy at nodes or on edges.

4.2.1 Microscopic Model Graph Requirements

In Chapter 3, we showed how Occupant Agents in EvacSim travel in two-dimensional space, directing their travel through manipulation of a two-dimensional vector which they use to steer towards their goal while avoiding obstacles. Routes produced

through path planning using graphs are converted into sequences of coordinate goals. A requirement of the navigation graph is that any edge between two nodes must not intersect with building geometry. This requirement guarantees that the agent can progress from goal to goal in its route plan without being impeded by building geometry.

While agents calculate route plans as a sequence of goals, they have the capability to observe the local area and determine if it is possible to jump forwards in their route plan if later goals in their route are within a clear line of sight. For instance, an agent at a position A might compute a route that takes it to B, C, D, E and F, but when it arrives at C it can see that F is within a clear line of sight from its current position, it can proceed directly to F from C (even if the graph node for F is not a direct neighbour of C), as shown in EvacSim in Figure 18.

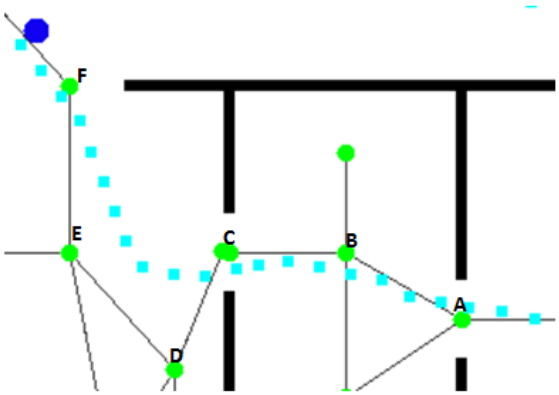


Figure 18: Agent skipping ahead on a route (EvacSim screenshot)

As the agents shorten paths by taking shortcuts when possible (discovered by looking ahead for clear line-of-sight to later goals in their route), the graph used by agents for planning should incorporate edges which are represent the likely shortcuts that would be taken. If such consideration is not made, then the agents will often over-estimate path lengths, and may make poor planning decisions as a result.

4.2.2 Macroscopic Planner Graph Requirements

EvacPlan is an implementation of the dynamic evacuation planner described by Hadciz [33], integrated with the EvacSim evacuation simulator. EvacPlan makes use of a network flow graph of the building, which is an extension to a traditional navigation topology graph that incorporates flow parameters on graph edges, describing the maximum number of occupants that can pass through the edge within a given time-frame and the amount of time taken for traversal of that edge. Dynamic, Flow-based Planning functions best when the graph features a low number of edges, ideally representing traversal between large spaces.

Large numbers of edges passing near to each other, or criss-crossing, mask the capacity constraints on occupants passing through the spaces and should be avoided. A further requirement is for discovery of node occupancy; each node should preferably cover a large amount of space so that occupancy information is represented in a manner suitable for evacuation planning algorithms (for example, single rooms should not be subdivided into multiple nodes as this hides the interaction of occupants from planners). Finally, the lower the graph complexity, the quicker the EvacPlan planner can compute optimised evacuation routes, a key consideration for timely emergency response.

As dynamic evacuation planners need to communicate evacuation routes (generated in terms of a path through the topological graph) to human users, a low detail graph is also useful for communication of routes to occupants. A simplified Room and Gateway graph would allow for natural descriptions of routes in terms of rooms and doorways to visit, and give an easy to understand high-level overview of the emergency status to safety personnel.

4.2.3 Hierarchical Detail levels

With these requirements in mind, we identify four levels of graph detail to be generated:

1. High Detail, dense "Movement graph" for agent movement
2. Initial "Fundamental Graph" which is used to generate other graph types
3. Simplified "Planner graph" for evacuation planning and long-distance agent planning
4. High-level "Structural graph" for communication of building state and evacuation routes to humans

4.3 Graph Generation Overview

In this section, we describe the use of the Fundamental Graph (algorithmically generated from two-dimensional building geometry) as a basis for building higher and lower detail graphs. With the Fundamental Graph as a common foundation, this facilitate translation between graphs of different detail levels. We convert this graph into a lower detail graph ("Planner Graph") through simplification techniques that reduce the detail of the graph. Additionally, from the Fundamental Graph we increase the graph detail to incorporate many extra nodes and edges, representing

the wide freedom of movement options available to agents moving in two-dimensional space (the "Movement Graph").

Finally we generate a Structural Graph from the Fundamental Graph by clustering nodes based on conceptual spaces bordered by doorways and entranceways. This Structural Graph amalgamates nodes in open spaces and allows for concise descriptions of routes in the building in terms of the traversal between gateway areas (e.g. doors) and the spaces they border. The simplification methodologies used maintain association between nodes at low-detail and high-detail, allowing for occupant agents to receive route instructions in terms of the Structural Graph, and convert this to Movement Graph routes easily. Similarly, agent locations establish node occupancy in the Fundamental Graph (which has a 1:1 relationship between graph nodes and building spaces, Appendix A.2.5), which can then be translated into occupancy data in the Planner Graph for the dynamic evacuation planner to use in evacuation planning. In this manner, we can use the most appropriate graph detail level for each separate task.

4.4 Fundamental graph generation

To generate other graph detail levels, we first generate the Fundamental Graph by dividing two-dimensional building geometry into discrete rectangular spaces ("Room" objects described in Appendix A). Each rectangle is represented by a graph node in the Fundamental graph, and the border region with adjacent rectangles is also represented by a graph node. Each Room is connected to its border nodes by an edge. We chose rectangular spaces as they are reasonably simple to generate and are concave, ensuring that all positions within a space have a clear line of sight to all other positions within that space. The purpose of the Fundamental Graph is to represent the rectangular subdivision and traversability of the building geometry in graph form, allowing the translation of two-dimensional building coordinate positions to nodes on the graph. With the Fundamental Graph acting as the basis for the other graph types, this allows the translation between various graph detail levels and the discrete space geometry of the building.

Generation of Rectangular spaces is performed by iteratively generating 1×1 Room objects and inflating them gradually until they come into contact with building geometry or existing spaces (Appendix A.2.4, Algorithm 13). Having covered the space of the building with Room objects, Intersections are generated in the border area between Rooms by inflating each Room 1 unit to the right and down, and identifying the 1-unit wide rectangular overlap (Algorithm 14). This overlap becomes an Intersection space. A Graph Node is labelled with an (x,y) coordinate position placed created at the centre-point of the Intersection node, and is connected by edges to

Graph Nodes at the centre of the two Rooms that produced the Intersection. In this manner, we cover the building space with a planar graph that guarantees that the lines connecting the coordinates associated with neighbour nodes do not intersect with geometry, and that the traversability of spaces is represented in the graph.

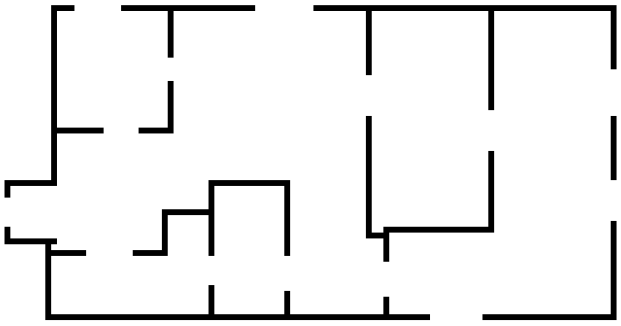
This approach to space division guarantees that each node is associated with a rectangular Room or Interesction (and hence, all the space represented by a node is visible and reachable from all other points in that space). The graphs produced are planar and the centrepoint of each space is visible from its neighbours (as the neighbours are intersection spaces).

This approach guarantees full coverage of the space without overlapping spaces. Graphs were generated using three building floorplans. The first is a small building (B1, Figure 19) featuring irregular rooms, with a large central hub and six outer doorways. This building produces graphs that feature a mix of node degrees and multiple path options. The second building (B2, Figure 20) features a grid-like layout, which produces many nodes with degree of four or greater, with the potential for a great deal of corner-cutting shortcuts. Finally we demonstrate the graph generation on a real-world building (Nimbus, Figure 21), based on the ground floor of the Cork Institute of Technology's Nimbus building. This building features a mix of long corridors and large spaces featuring central pillars, as well as a limited number of outer exits.

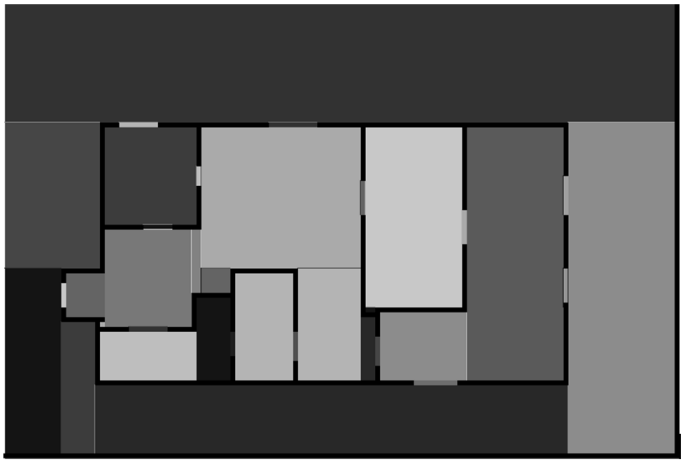
4.4.1 Fundamental Graph

The Fundamental graph generation method was performed on the three buildings to characterise the graph complexity and detail produced for these different cases; the graph complexity is shown in terms of node count (Figure 22) and edge count (Figure 23). In these results we see that the graph node count rises with the irregularity of the building structure, as well as with the number of walls. We also see that the node degree rises with this increased irregularity; the subdivision of irregular spaces produces Room objects with many neighbours.

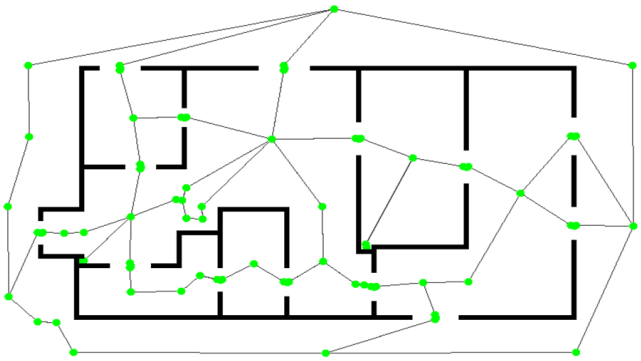
To evaluate the usefulness of the graph for simulated pedestrian path planning, occupants agents were generated one at a time, in randomised positions in the building. Each agent is given the goal of travelling to a randomised destination (routes planned using A* algorithm). The average path-planning time is recorded in each case, as well as the ratio between estimated path length on the graph and the real distance travelled by the agents following the routes. Dividing the estimated path length by the actual distance travelled, and averaging, gives the Path Length Accuracy (Figure 24), with values close to 1 indicating accurate estimates and high values indicating that the graph overestimates the distance that would be travelled



(a) B1 Floorplan



(b) B1 Room Generation

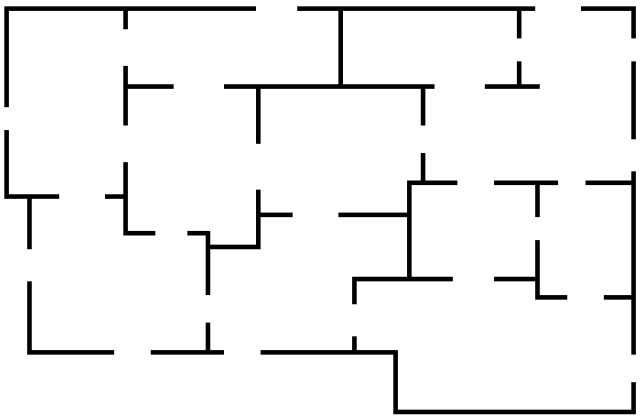


(c) B1 Fundamental Graph

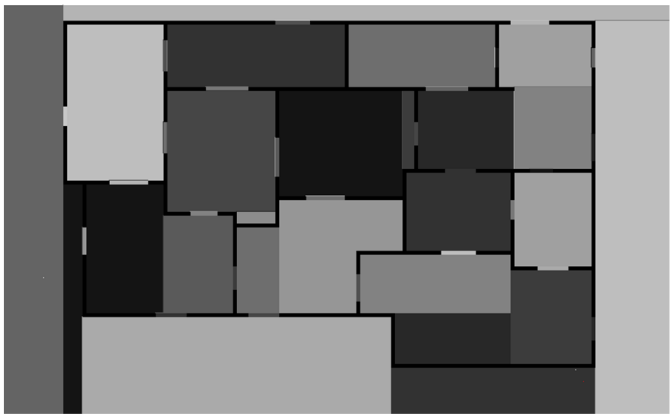
Figure 19: Fundamental Graph and Rooms (B1)

by the agents.

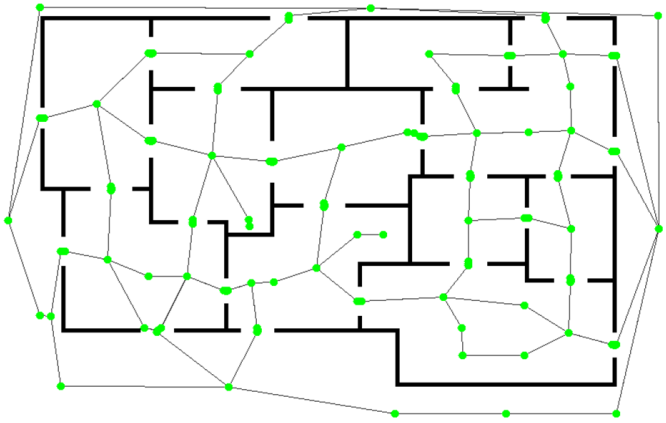
In these results, we observe that the path length of the routes produced using A* on the Fundamental Graph tend to overestimate the distance the agents travel by a significant factor. In B2, there were many opportunities for cutting across corners, and as such, the distances travelled by agents was, on average, almost half the distance estimated by the graph. While the computation time for this graph detail is reasonably low, this underestimation of path length can cause agents to make



(a) B2 Floorplan



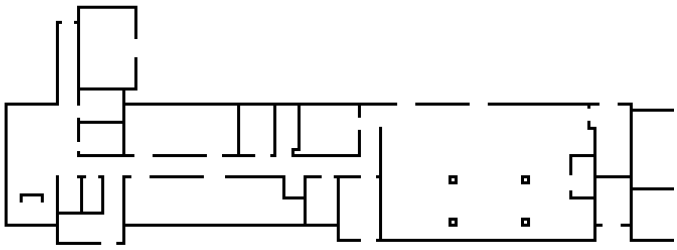
(b) B2 Room Generation



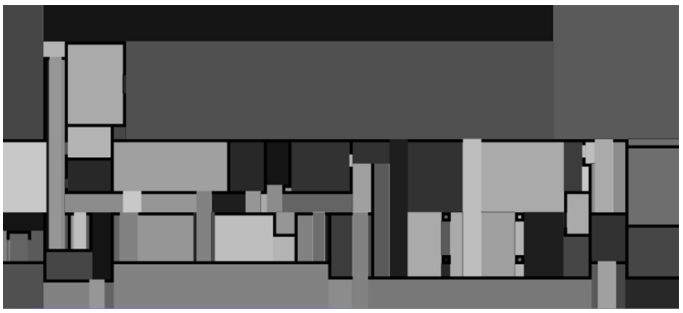
(c) B2 Fundamental Graph

Figure 20: Fundamental Graph and Rooms (B2)

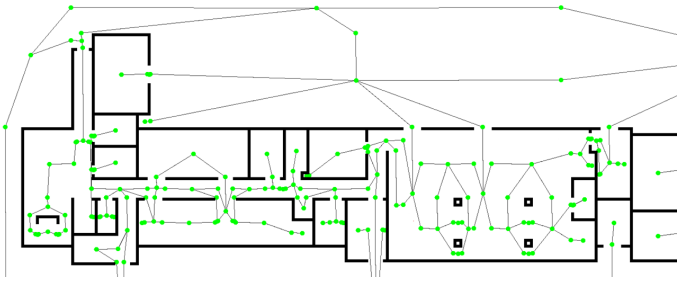
poor path-planning decisions such as taking long routes that merely appear shorter than alternatives on the graph. This suggests that high-detail graphs with more nodes and edges might be more appropriate for agent movement planning than the fundamental graph alone.



(a) Nimbus Floorplan



(b) Nimbus Room Generation



(c) Nimbus Fundamental Graph

Figure 21: Fundamental Graph and Rooms (Nimbus)

4.5 Graph Simplification (Planner Graph)

For some building geometries, the Fundamental Graph can feature a great many nodes and edges, which can lead to excessive computation times for dynamic evacuation planning and the large number of short edges does not translate well into a network flow graph. To reduce the complexity of the graph (producing a Level 3 Graph suitable for evacuation planners) we employ two simplification techniques to group nodes together: Loop Removal and Triangle Removal.

4.5.1 Loop Removal

Loop Removal is a method by which adjacent nodes are recursively merged together if they each have a degree of two and are each a neighbour of the other, except in the case that the merged node would not have line of sight with the neighbours of the original nodes (Algorithm 1). These nodes arise as a result of small adjustments in the dimensions of spaces that are approximately rectanglar (such as a corridor that

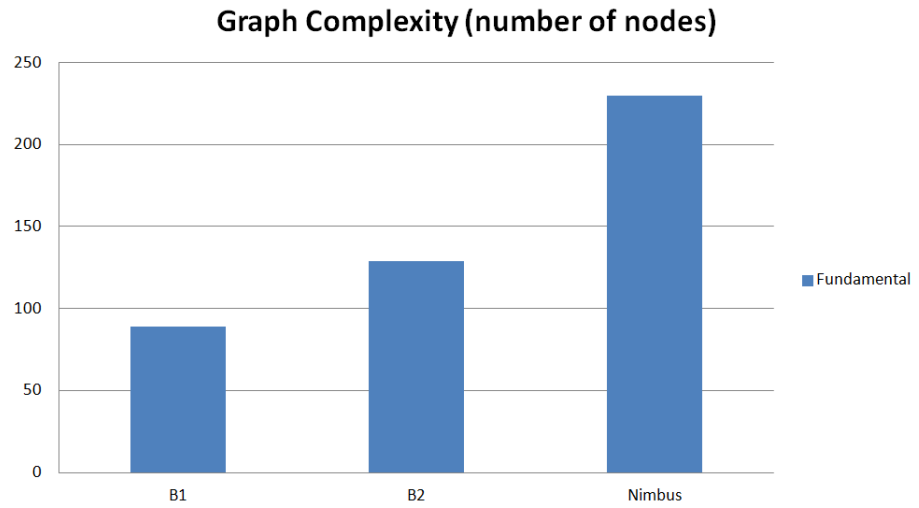


Figure 22: Graph Complexity (node count), fundamental graph

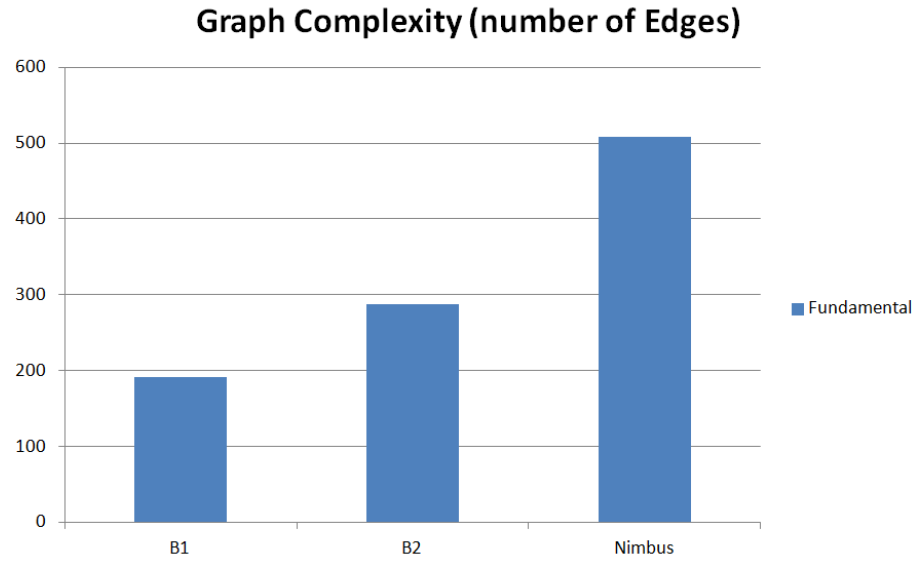


Figure 23: Graph Complexity (edge count), fundamental graph

widens slightly in one part), or at doorway spaces where clusters of three nodes arise representing the doorway itself and the Intersection Nodes connecting the adjacent Rooms.

The effect of this simplifcation is that a series of two-degree nodes in sequence become merged together, a useful result for simplification of long corridors or large spaces. An additional result is that closed loops on the graph become absorbed into the graph or become reduced to individual nodes (Figure 25).

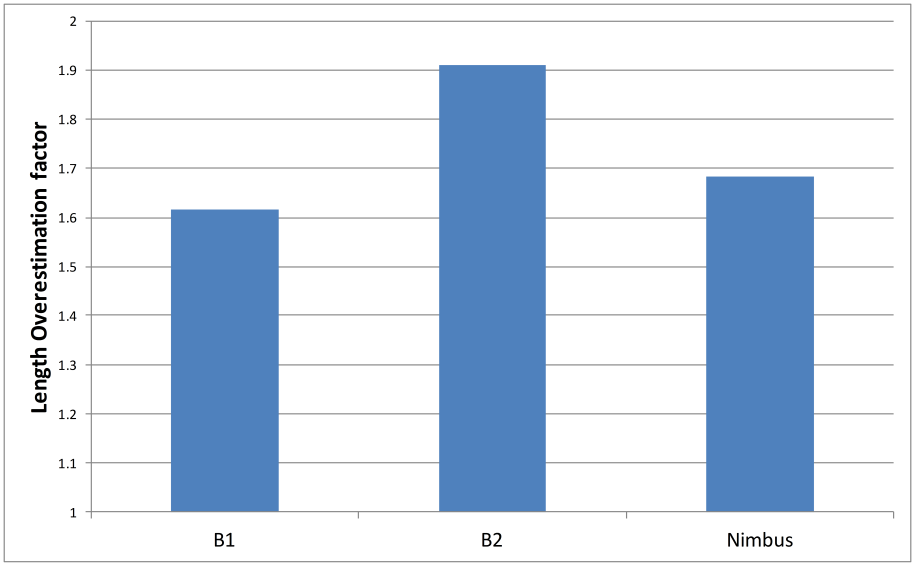


Figure 24: Path Length Accuracy for Fundamental Graph

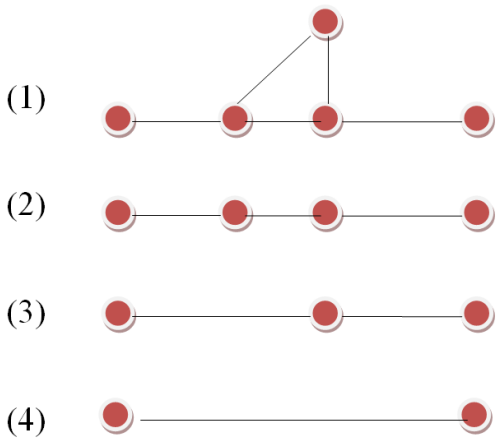


Figure 25: Loop Removal Graph Simplification

Algorithm 1: Loop Removal

```
Data: (Graph) g
(boolean) reduced = true;
while reduced do
    reduced = false;
    foreach Node n ∈ g do
        if n.degree == 2 then
            (Node) a = n.neighbours[0];
            (Node) b = n.neighbours[1];
            if hasLineOfSight(a,b) then
                g.remove(n);
                g.connect(a,b);
                reduced = true;
            end
        end
    end
end
```

4.5.2 Triangle Removal

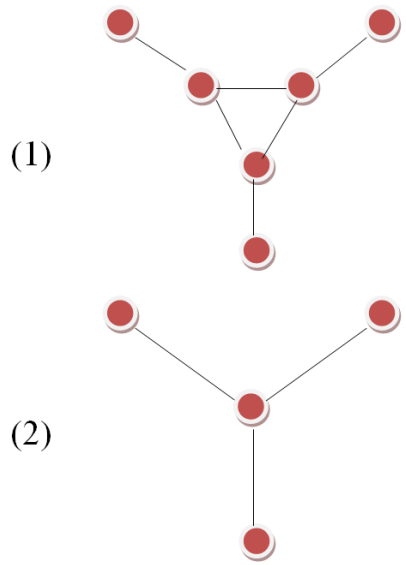


Figure 26: Triangle Removal Graph Simplification

The second method for graph simplification at this level is to recursively merge triangle groups together. These node groups feature triplets of nodes that are each connected to the other, which allows for more nuanced agent path-planning but would be better represented by a simple junction structure (Figure 26) in order to capture the potential interaction of flow units in flow-based evacuation planning. To identify these triangular groups, we search the graph for nodes with degree of three. When such a node is identified, we check its neighbours to see if they any are also neighbours of each other. If this is the case, then the nodes can be merged together (Algorithm 2), provided that the edges connecting to the merged node do not violate the line of sight with other nodes in the graph (i.e. edges cannot pass through walls).

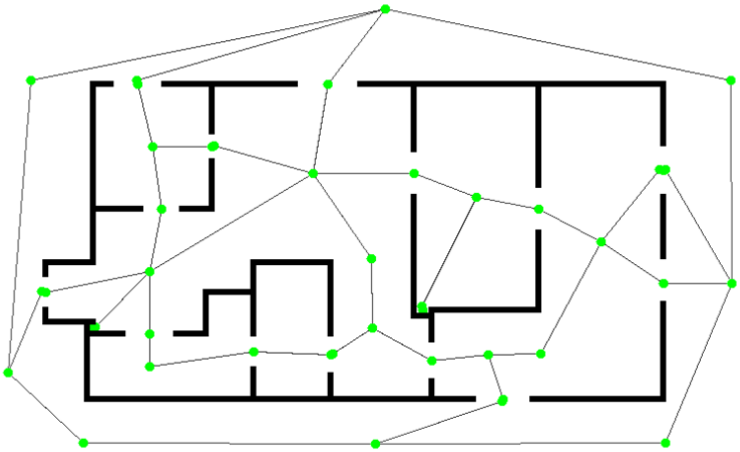
Algorithm 2: Triangle Removal

```
Data: (Graph) g
(boolean) reduced = true;
while reduced do
    reduced = false;
    foreach Node n ∈ g do
        if n.degree ==3 then
            ((Node) a = n.neighbours[0];
            (Node) b = n.neighbours[1];
            (Node) c = n.neighbours[2];
            (Node)common = null;
            if g.connected(a,b) then
                if g.connected(a,c) then
                    common = a;
                end
            else if g.connected(a,c) then
                if g.connected(c,b) then
                    common = c;
                end
            else
                g.connected(b,c)
            end
            if top.connected(a,b) then
                common = b;
            end
            if common != null then
                g.remove(n);
                reduced = true;
            end
        end
    end
end
```

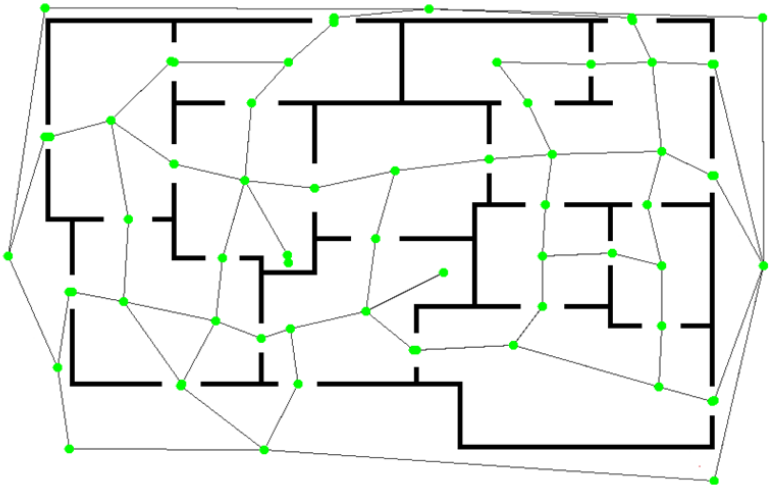
Applying Loop and Triangle Removal to the Fundamental Graphs generated in Section 4.4 produces the simplified Planner Graphs shown in Figure 27. These graphs feature substantially fewer nodes and edges than the original Fundamental Graph (Figures 28, 29).

4.5.3 Planner Graph Simplification Results

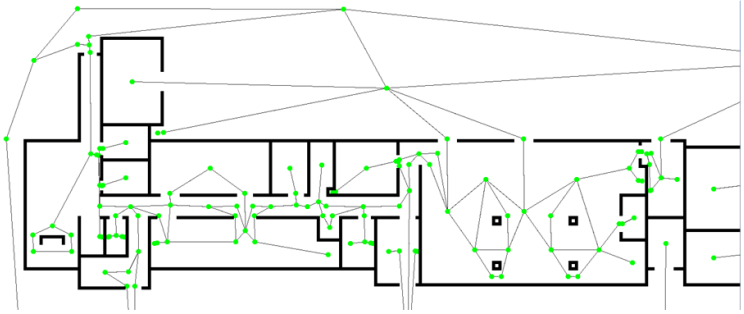
Applying loop removal and triangle removal to the Fundamental Graphs produced in Section 4.4 leads to a substantial reduction of graph complexity, giving longer edges and fewer nodes. This simplified graph preserves large-scale navigation detail and helps ensure that network flow graphs produced from this graph do not mask congestion effects through excessive graph detail. These graphs feature a reduced node (Figure 28) and edge count (Figure 29) relative to the original fundamental



(a) B1 Planner Graph



(b) B2 Planner Graph



(c) Nimbus Planner Graph

Figure 27: Planner Graphs (Loop and Triangle Removal)

graph.

The Planner graphs were converted into network flow graphs for use with the EvacPlan dynamic evacuation planner by allocating flow parameter values and traversal time values to edges in the graph (this process of discovering appropriate flow values to assign to edges is described in detail in Chapter 5) . Occupant and hazard

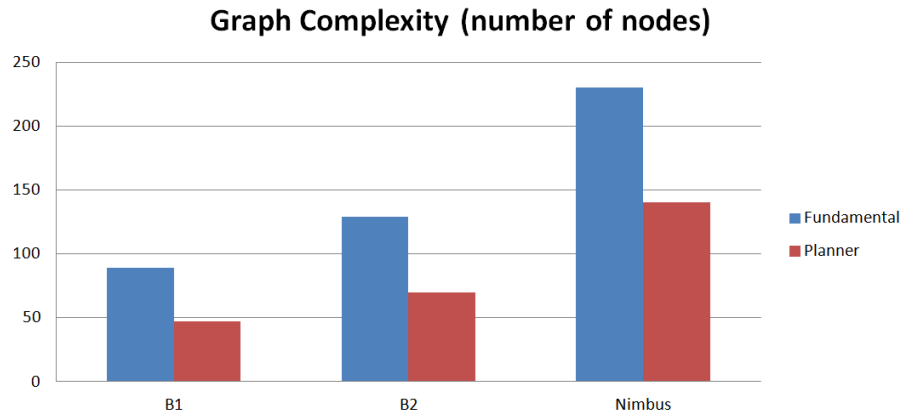


Figure 28: Planner Graph Complexity (node count)

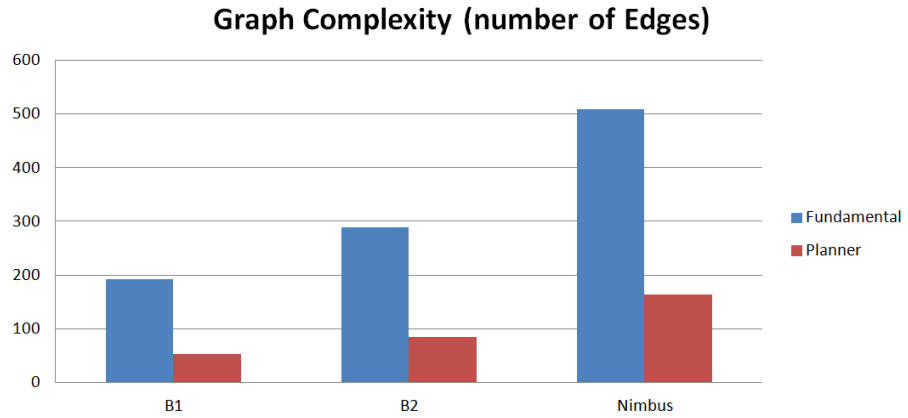


Figure 29: Planner Graph Complexity (edge count)

locations are randomized and EvacPlan computes optimized evacuation routes for the occupants, with the average computation time recorded in each case, shown in Figure 30. In these results, we find that the Planner computation times are very low and well within the time constraints of emergency response situations. Further investigation into dynamic evacuation planner performance using coupled Flow Graphs based on the Planner Graphs generated here can be found in Section 5.7.

The simplified Planner Graph was used for Agent route planning, again using the A* algorithm [8]. This simplified graph results in somewhat lower path-length overestimation than the fundamental graph (Figure 31), as it has a general "smoothing" effect on graph features that produced "zigzag" paths, by eliminating superfluous nodes along an otherwise straight path. While the accuracy is improved, there are still short-cuts to be taken, particularly in the mesh-like structure of B2 (Figure 33(b)) as agents opt not to travel all the way to the centre of Room Nodes while traversing from one doorway to another at an angle.

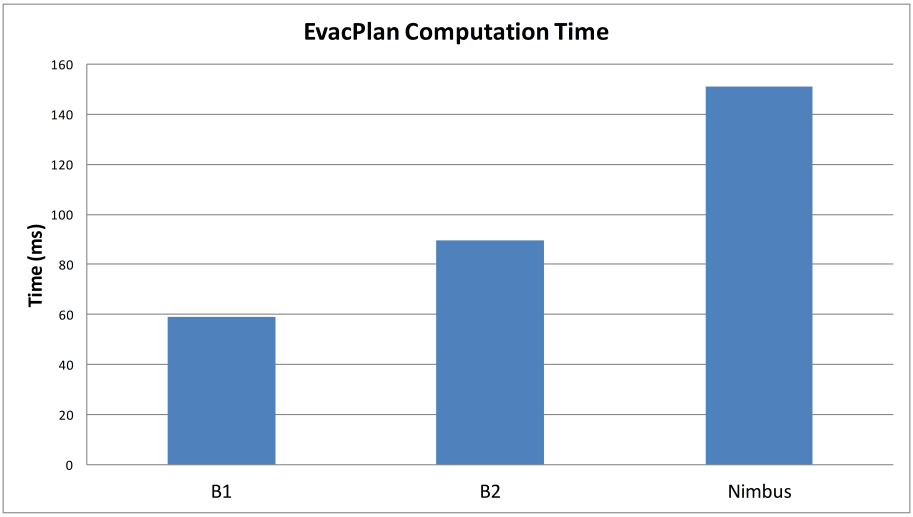


Figure 30: EvacPlan running time using Planner Graphs

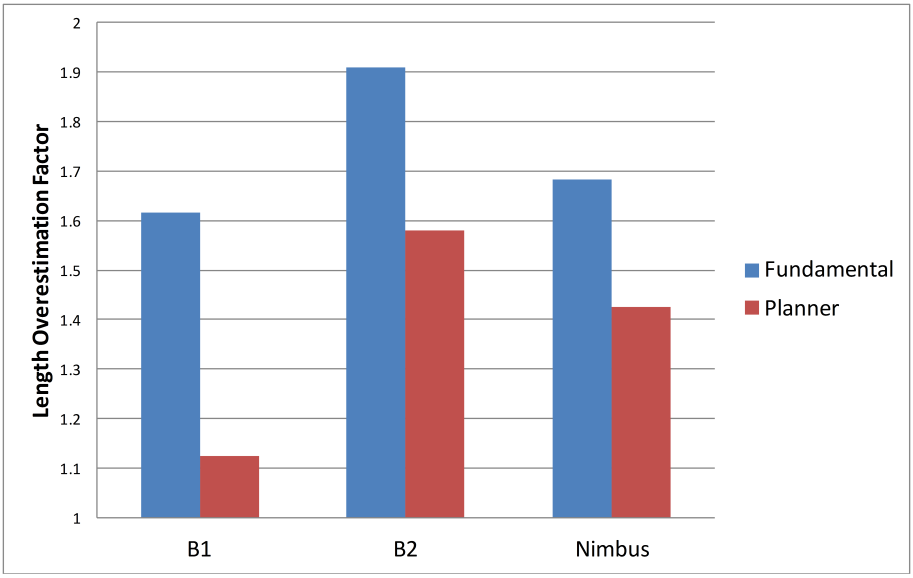


Figure 31: Path Length Accuracy, with Planner Graphs

4.6 Increasing Detail (Movement Graph)

As the agents take short-cuts in movement, accurate agent path planning requires nodes and edges that represent these shortcutting paths. While the Planner Graph showed an improvement in Path Length Accuracy for A* planning relative to the fundamental graph, we found that there were still several short-cuts being taken by agents due to the relatively sparse and rigid graph structure. More accurate path length estimates could be achieved by representing likely short-cuts on the graph. To achieve this, we introduce additional nodes and edges on the graph: Nodes are added near each convex corner in the building geometry, and connected to the Room space in which they are contained (Algorithm 3).

Having added these Corner Nodes, we then connect each Corner Node to any other Corner Node for which there is a clear line of sight (Figure 32). This increases the

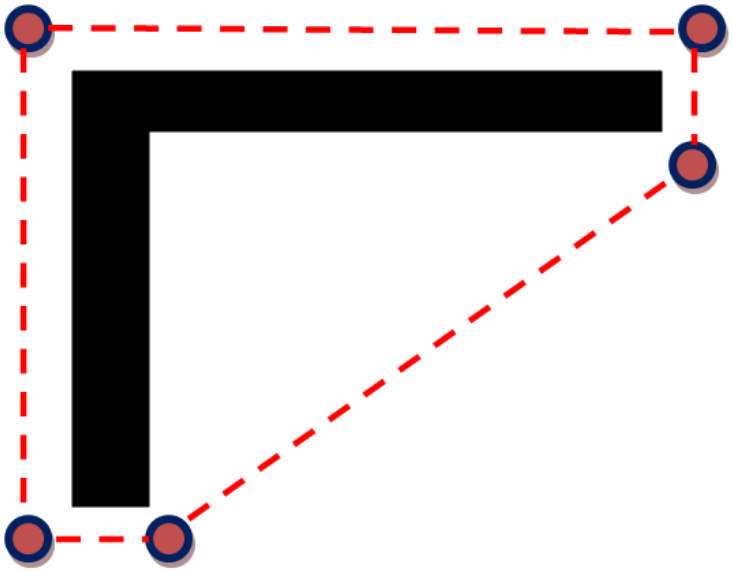


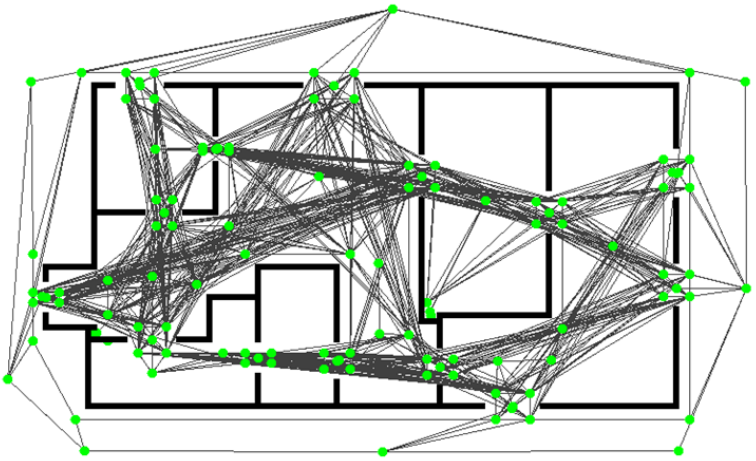
Figure 32: Adding Corner Nodes on convex corners

complexity of the graphs, but allows for paths that cut close to corners and skip across spaces in a single edge that would require multi-hop paths on other graph detail levels (which would be skipped over by the shortcut-taking agents).

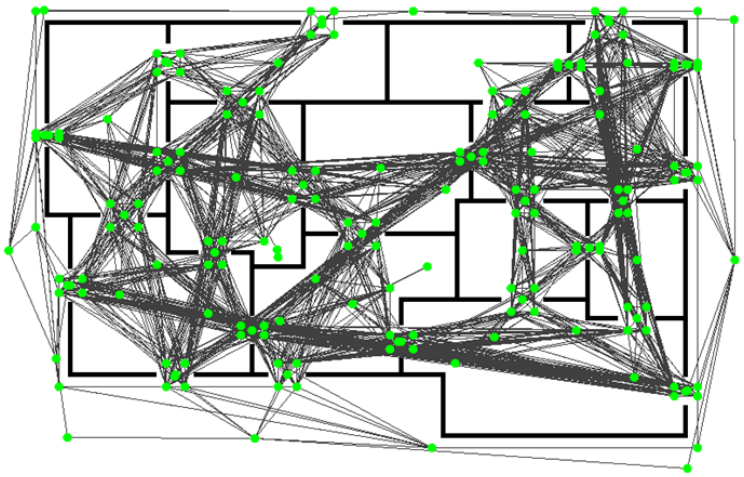
Algorithm 3: Corner Node Generation

```
Data: (Graph) g, (Set) Walls, (Set) Rooms
(Set) CornerNodes;
foreach Wall w ∈ Walls do
    (Coordinates) TL = w.getTopLeft;
    TL.x = TL.getX -4;
    TL.y = TL.getY -4;
    if !clipped(TL,Walls) then
        CornerNodes.add(Node at TL);
    end
    (Coordinates) TR = w.getTopRight;
    TR.x = TR.getX +4;
    TL.y = TL.getY -4;
    if !clipped(TR,Walls) then
        CornerNodes.add(Node at TR);
    end
    (Coordinates) BL = w.getBottomLeft;
    BL.x = TL.getX -4;
    BL.y = TL.getY +4;
    if !clipped(BL,Walls) then
        CornerNodes.add(Node at BL);
    end
    (Coordinates) BR = w.getBottomRight;
    BR.x = BR.getX +4;
    BR.y = BR.getY +4;
    if !clipped(BR,Walls) then
        CornerNodes.add(Node at BR);
    end
    foreach Node n ∈ CornerNodes do
        Node location = g.getNode(whereIsThis(n.coordinates);
        graph.add(Edge(location,n));
        foreach Node m ≠ n ∈ CornerNodes do
            if !∃(Wall) between n,m then
                g.add(Edge(n,m));
            end
        end
    end
end
```

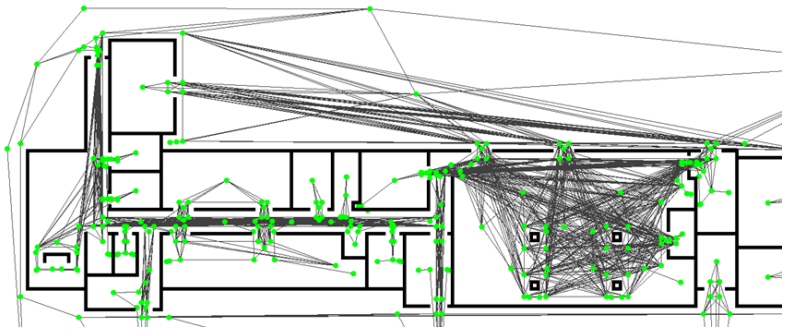
As we can see in Figure 33, the additional nodes and line-of-sight edges causes a great deal of edge generation in open areas that feature multiple entranceways. This is especially apparent in the mesh-like structure of the B2 building, where most rooms have three to four entrances. The space containing columns to the right-hand side of the Nimbus building also produces a dense network, as a result of incorporating corner nodes and associated edges around each of the columns. The impact of this increased complexity is shown in terms of Node Count (Figure 34) and Edge Count (Figure 35).



(a) B1 Movement Graph



(b) B2 Movement Graph



(c) Nimbus Movement Graph

Figure 33: Movement Graphs (Convex Corner Nodes)

As these high-detail graphs are intended to be used for agent movement planning, the A* path planning results are of particular importance. In Figure 36 we show the Path Length Accuracy result using the Movement Graphs, which demonstrates that paths chosen on the Movement Graphs correspond closely to the actual paths taken by the occupant agents. These results indicate that the Movement graph is

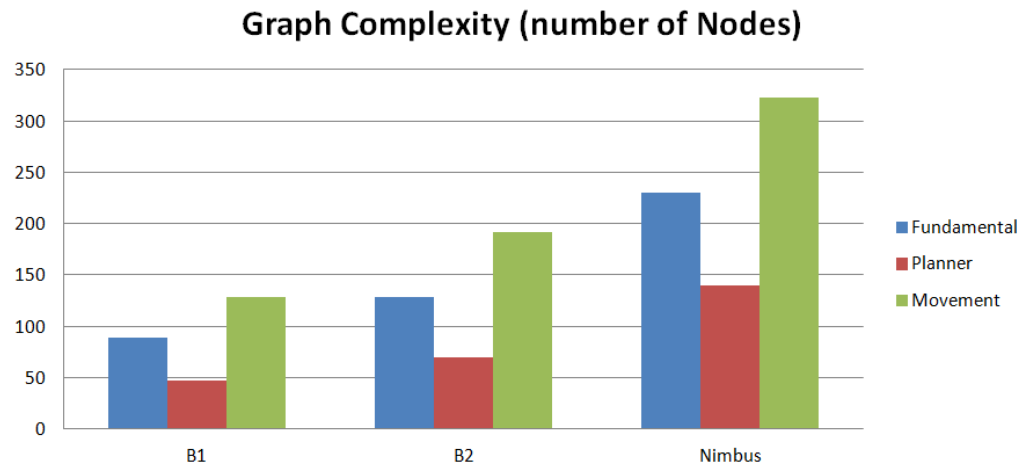


Figure 34: Movement Graph Node Complexity

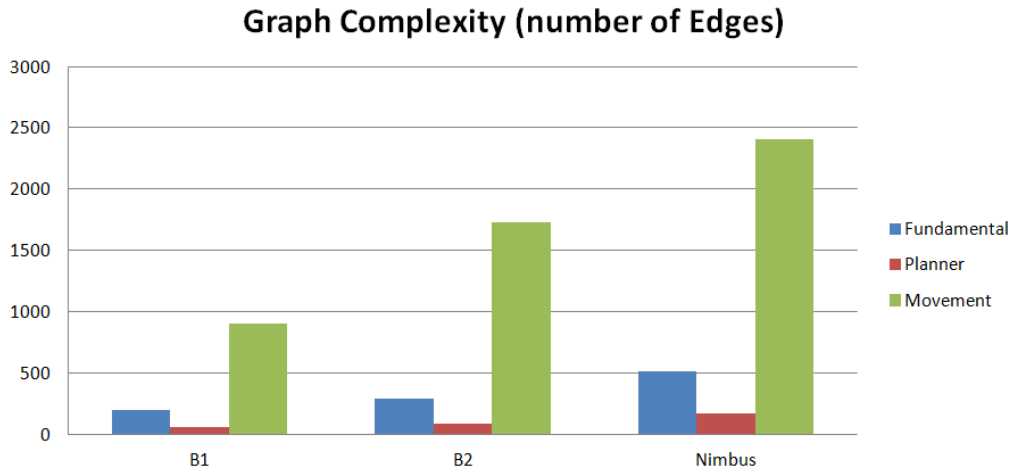


Figure 35: Movement Graph Edge Complexity

an excellent choice for high-detail, local area movement planning, though at greater computational cost than using lower detail graphs (cf. Section 4.8).

4.7 Structural graph simplification

A high-level Structural Graph provides a low-detail model of the building appropriate for communication of routes to occupants, and for relaying of evacuation status to safety personnel. This graph is generated according to the principles described by Richter et al [30] by dividing the building at doorways and other exits, known as "gateway nodes". Gateway nodes can be identified in the building geometry definition (such as door objects in Industry Foundation Classes building models, Section 2.3) or discovered algorithmically, as detailed in Appendix A.2.4.

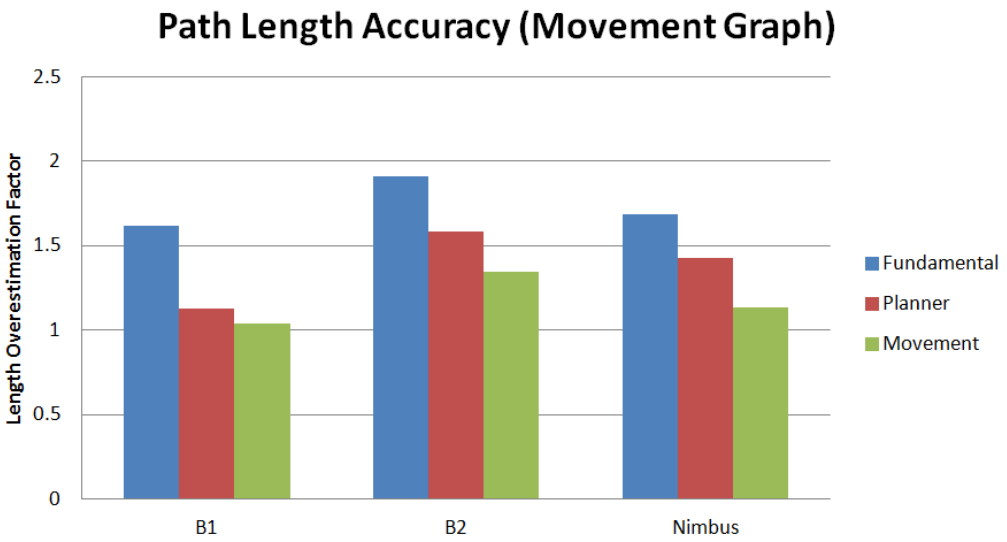


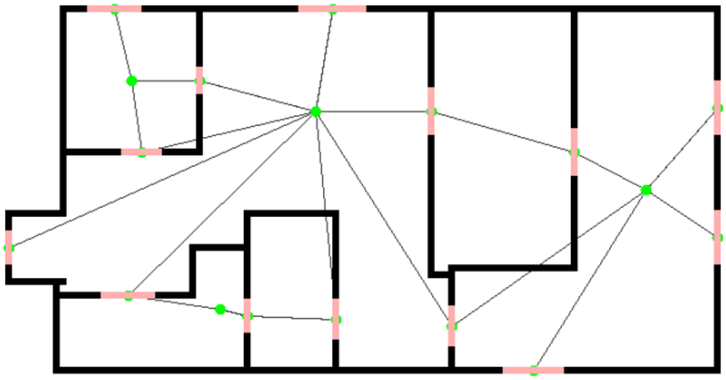
Figure 36: Path Length Accuracy (Movement Graph)

Algorithm 4: Structural Graph Generation

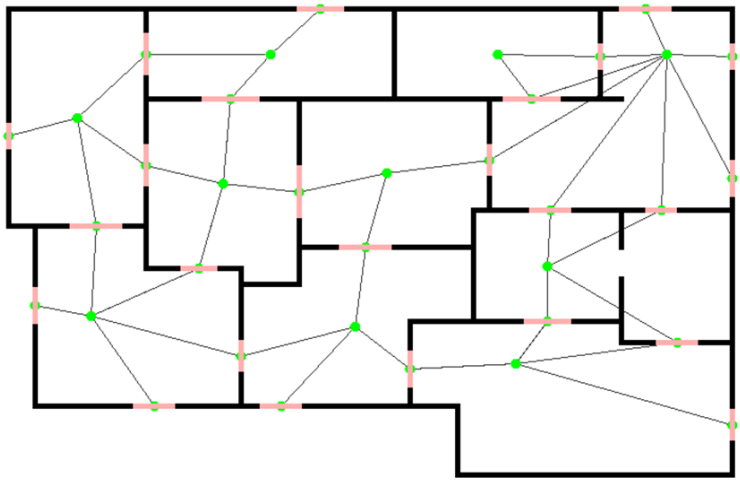
```
Data: (Graph) g, (Set) Gateways)
Result: (Graph) S
(Graph) S = new Graph;
(boolean) reduced = true;
while reduced do
    reduced = false;
    foreach Node n ∈ g do
        if n ∋ Gateways then
            merge(n,(n.neighbours ∋ Gateways));
        end
    end
end
```

To generate the Structural Graph, we consider two types of graph node: Gateway (doorways) and Space (everything else). We merge nodes so that all the neighbours of a Space node are Gateway nodes, and vice versa. We achieve this by recursively merging non-Gateway nodes and their non-Gateway neighbours until no new merges are performed. The result of this clustering is that the graph becomes divided up into large, irregular discrete spaces represented by a single Space Node. Traversal between Spaces always takes the form of a transition from Space to Gateway, to Space etc. The result of applying this simplification routine to the building graphs from Section 4.4 is shown in Figure 37.

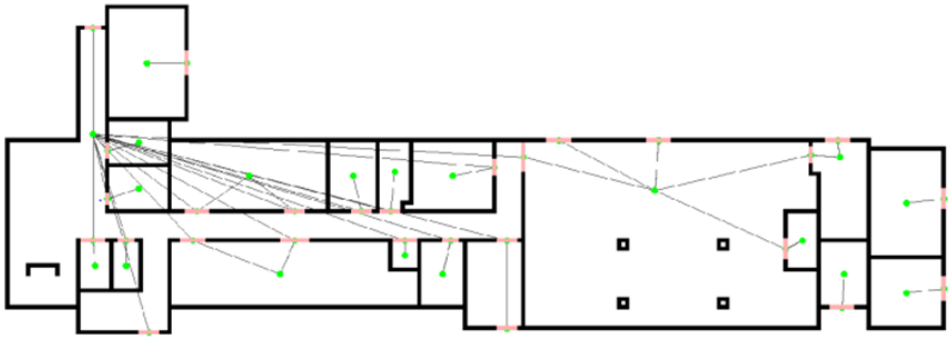
These graphs are substantially less complex than those at higher levels of detail, and routes in the graph can be explained in simple terms of transitions between large spaces and the gateways between them. As all paths in the Structural Graph follow the Pattern of *"Space-Gateway-Space-Gateway-Space-Gateway-Space"*, they can be expressed as a series of Gateways. With labelled nodes, these paths could easily be described in sequence (e.g. exit through the East Door, to enter Corridor 2, then



(a) B1 Structural Graph



(b) B2 Structural Graph



(c) Nimbus Structural Graph

Figure 37: Structural Graphs (merging nodes behind Gateways)

to the Front-Lobby-Entrance). The node and edge count for Structural Graphs is given in Figure 38, illustrating the simplification possible with this approach.

An additional utility of these Structural Graphs is that it allows for the easy discovery of the "outside" space of the simulation world, which allows for the identification

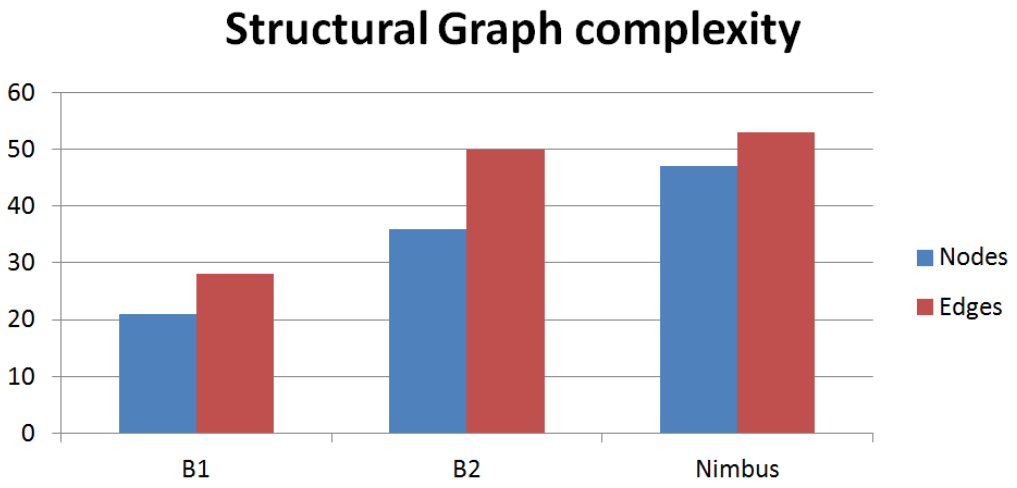


Figure 38: Structural Graph Complexity

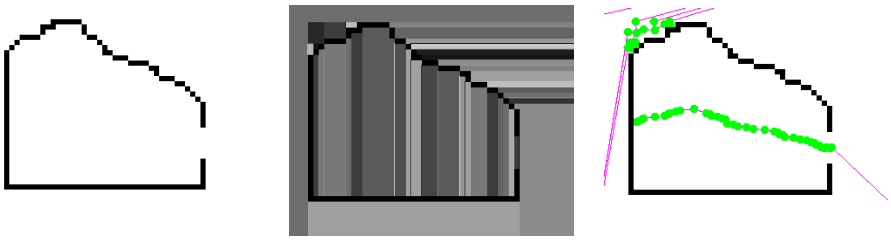
Table 2: Graph Simplification results for irregularly shaped building

	Nodes	**Edges**
Fundamental Graph	111	260
Planner Graph	28	64
Structural Graph	3	2

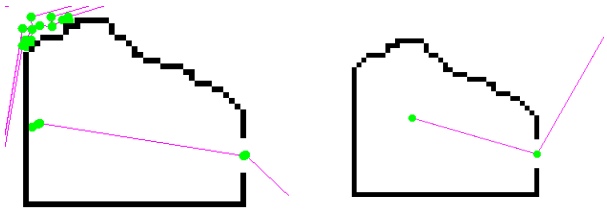
of arbitrary positions as being "inside" or "outside", and also allows for algorithmic identification of building Exits, as all neighbour nodes to the "Outside Node" are by necessity, the last Gateway nodes before reaching "outside" space.

As all the empty space surround the building is amalgamated into a single Non-Gateway node, performing "WhereIsThis" (Appendix A.2.5) on Coordinates (1,1) will return the Room object for the top left empty space. As this Room is a subnode of the non-Gateway node that surrounds the building, the WhereIsThis call will return the node that represents the outside space ("Outside Node"). From this, we can then determine which nodes represent the final exits of the building by simply noting all the neighbours of this Outside Node. We can also determine whether arbitrary coordinates are "inside" or "outside" if calling the WhereIsThis method on them returns a subnode of the Outside Node.

To illustrate the impact simplification can have, we show in Figure 39 the result of applying loop and triangle removal, and Gateway Clustering to an irregularly shaped building (which begins with many narrow room objects, and hence, superfluous graph nodes on the fundamental graph), with the node and edge counts for these graphs shown in Table 2.



(a) Floorplan (b) Rooms (c) Fundamental Graph



(d) Planner Graph (e) Structural Graph

Figure 39: Graph simplification on an irregularly shaped building

4.8 Graph Complexity (A* Running Time)

In generating varying levels of graph detail in an interconnected manner, we produced varying results for relative complexity and accuracy for different tasks such as agent movement or dynamic evacuation planning. The different building types showed the impacts of various building structure features on the resulting graphs, such as the mesh structure found in the Grid-like B2 when the Movement graph is generated. Such features can have a significant bearing on the computational complexity and accuracy for planning on these graphs. The running time (in nanoseconds) for A* on the Fundamental, Planner and Movement Graphs during the Path Length Accuracy experiments is given in Figure 40, averaged for 800 randomised agents/paths.

While the Movement Graphs performed well on the Path Length Accuracy metric (Section 4.6), the increased running time for Movement Graph path planning relative to the Planner Graph suggests that there may be gains to be made by combining both approaches; using a Planner Graph for long-distance planning to produce a set of intermediate waypoints and using the Movement Graph for the short distance planning between these waypoints (Section 4.9).

4.9 Hybrid Path Planning for Agents

In Sections 4.4 and 4.6 we showed simulated occupant agents making use of the fundamental and movement graphs for path planning (using the A* algorithm). These graphs allowed agents to navigate the space and in the case of the Movement graph in Section 4.6, paths on the graph corresponded closely to the actual paths

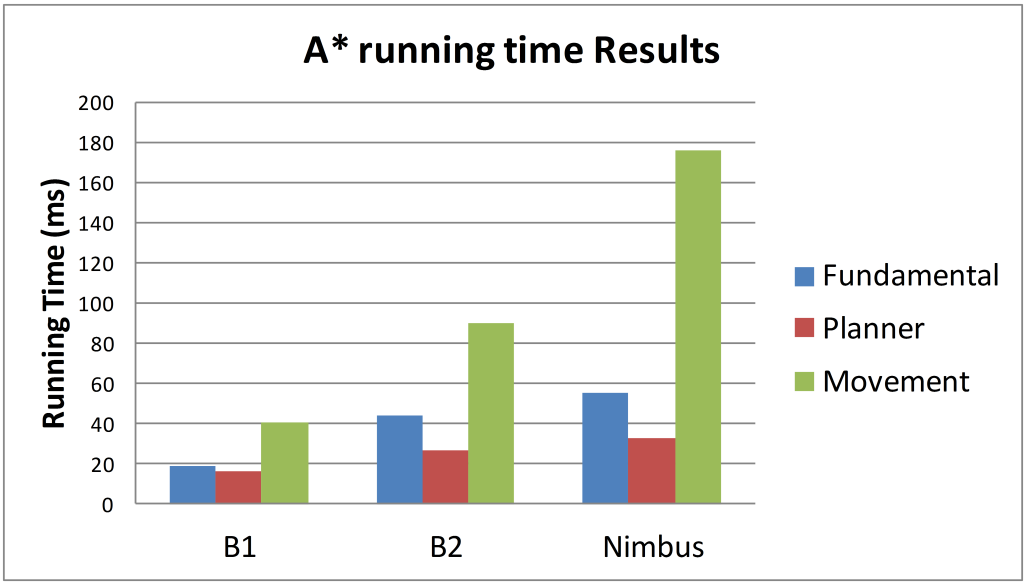


Figure 40: Summary of A Planning Computation Time results*

followed by agents moving from node to node.

In Section 4.5 we described a method for simplifying the graph to accommodate the requirements of dynamic evacuation planners. As there is a hierarchical association between the Fundamental and Movement Graphs (as each node in the Fundamental graph is also present on the Movement graph, and any additional corner nodes are connected to the graph based on their location in the building), we can perform high-level planning on Planner Graphs while following Movement Graphs for local movement (Algorithm 5) and support the provision of guidance generated by the dynamic evacuation planner in terms of paths on the Fundamental Graph to agents which then navigate the paths by using the movement graph, using steps in the Planner route as intermediate waypoints on the more detailed Movement Graph, Algorithm 6.

Algorithm 5: Hybrid Path Planning

```
Data: MovementGraph mg, PlannerGraph pg, Node start, Node end, agent
      A
(Node) currentLocation = whereIsThis(A.coordinates);
(Queue) proute = pg.getShortestPath(start,end);
while currentLocation ≠ end do
    (Queue) mroute = shortestPath(currentLocation, proute.nextNode);
    A.travelRoute(mroute);
    currentLocation = next;
end
```

Algorithm 6: Provisioning dynamic evacuation planner routes to agents

```
Data: MovementGraph mg, PlannerGraph pg, Route proute, agent A
while currentLocation ≠ end do
    (Node) next = proute.pop();
    (Queue) mroute = mg.getShortestPath(currentLocation, next);
    A.travelRoute(mroute);
    currentLocation = next;
end
```

As routes are given to agents in terms of the Planner Graph, but are actually followed using the Movement Graph (with route look-ahead behaviour), we can expect that the routes taken by the agents will differ in length from the routes presented by the dynamic evacuation planner. While the dynamic evacuation planner route optimizes routes for groups on the Planner graph, the microscopic agent behaviour combined with the greater detail of the Movement Graph results in the distance travelled results shown in Figure 41. Here we find that the agents perform a small amount of short-cutting between waypoints but the distances suggested by the path length are very similar to that travelled by agents (averaged for 800 randomised agents/paths).

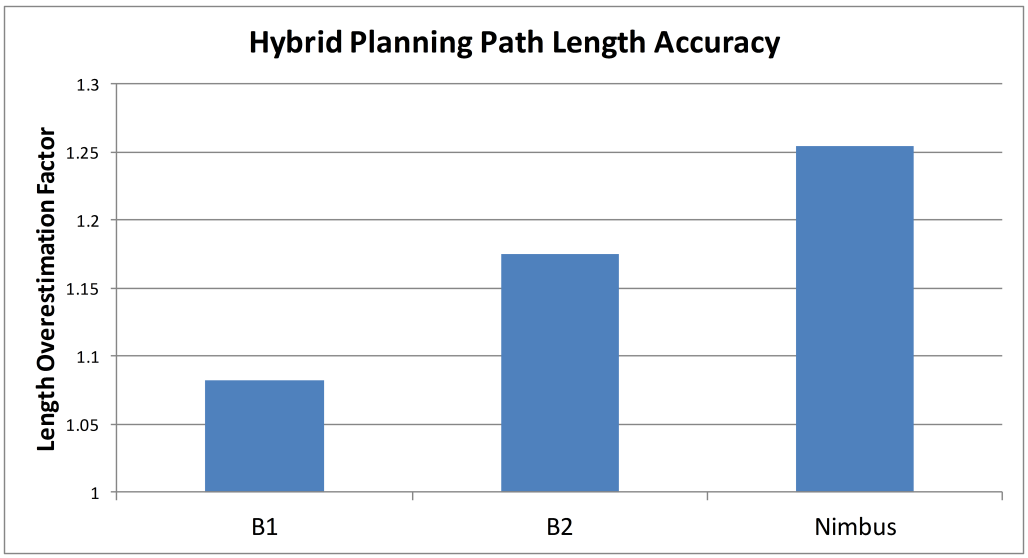


Figure 41: Path Length Accuracy (Hybrid Planning using plans provisioned by dynamic evacuation planner)

4.10 Conclusions

In this Chapter, we described the generation of low-complexity Planner Graphs for dynamic evacuation planners, and dense high-detail Movement Graphs for Agent Movement. These two graph detail levels are relatable to each other by way of the initial Fundamental Graph which is generated algorithmically based on the building geometry. We also demonstrated the use of both Movement and Planner Graphs through "Hybrid Agent Planning", allowing for paths described on the Planner

Graph to be given to occupant agents which can then follow them using the Movement Graph, showing the capability of agents to receive EvacPlan routes in terms of the Planner Graph and enact them using higher detailed local movement decisions using the Movement Graph.

In this Chapter, we also introduced a method for generation of very sparse "Structural Graphs", which again are derived from the original Fundamental Graph but feature greatly simplified structure representing the high-level description of building topology by dividing space into large compound spaces. This approach allows for routes generated on the Planner Graph to be described in terms of the compound spaces they pass through on the Structural Graph, providing the capability of natural language description of routes in terms of Rooms and Exits. This Structural Graph also provides the capability of identification of building perimeter exits, and whether a position is "inside" or "outside" of the building.

In Chapter 5, we investigate discovery of Network Flow Parameters for graph edges on the Planner Graphs generated in this Chapter. These coupled graphs are then used in the detailed evaluation of a Dynamic Evacuation Planner in Section 5.7.

5 Micro/Macro Coupling of Flow Characteristics in Network Flow Graphs for Prediction and Planning

5.1 Introduction

Dynamic evacuation planners (cf. Section 2.4.3) compute optimal evacuation routes for building occupants in emergencies by reasoning about the flow characteristics of the building in terms of a Network Flow Graph. Network Flow Graphs are graph models of building space representing the traversability between areas, and represent the maximum throughput of occupants and traversal time as parameters on the graph edges. For these dynamic evacuation planners to provide reasonable and safe plans, the flow parameters need to accurately reflect the reality of the building space, a task traditionally performed by hand with the use of building fire safety guidelines such as the Irish Technical Guidance for Fire Safety [84].

While using guideline rules to establish flow parameters can be useful, it is also time-consuming and may miss subtleties of the building structure impacting on flow. In Chapter 4 we investigated algorithmically generating building topology graphs for building geometries, but these graphs do not feature flow parameter information. By iteratively simulating crowd movement on each of the edges in the Planner Graph using EvacSim, we can determine realistic values for maximum achievable throughput, as well as lower and upper bounds on traversal times on the edges for high-traffic and low-traffic scenarios. These figures can then be assigned to the edges, modelling the flow parameters of the graph, a "Network Flow Graph" [29].

In this chapter we show use of the microscopic to macroscopic coupling approach, and perform experiments in the buildings used in Chapter 4 to determine if a per-edge coupling accurately represents the flow characteristics for multi-edge paths. The experiments in this chapter demonstrate the utility of macroscopic Coupled Flow Graphs as predictive tools to determine the throughput and travel time taken for groups of occupants in the building to travel arbitrary distances in microscopic simulation. Furthermore, these Coupled Flow Graphs provide the foundation for Dynamic Planners to generate congestion-sensitive evacuation strategies, which we utilise in Section 5.7.

5.2 Network Flow Graph parameters

In Chapter 4 we discussed the generation of several levels of graph detail modelling traversability of building space. One of these graph detail levels, the "Planner

Graph" featured a low number of nodes and edges. To allow for flow-based path planning, the edges of this Planner Graph are augmented with the flow parameters of Capacity, minTraversal and maxTraversal.

- Capacity: Optimum rate of agents per time unit traversing the edge
- minTraversal: Shortest time taken for a single agent to traverse the edge
- maxTraversal: Time take for Optimum rate of agents to traverse the edge

As the impact of building geometry and obstacles can cause flow in one direction to differ to that in another, the graph edges are bi-directional (implemented using directed edges pointing in both directions, which are coupled separately). Using traversal times, the graph can be used for:

- Shortest path planning using algorithms such as Dijkstra's algorithm or A*
- Maximum safety path planning, avoiding proximity to a hazard

When the number of occupants travelling across multiple edges is known, the minimum ("min") and maximum ("max") traversal times can be combined with the Capacity values for the edges to predict the total traversal time taken for groups along paths, which we investigate in Section 5.5.

5.3 Flow Coupling

In Chapter 3 we showed that the EvacSim pedestrian model provides a realistic model of pedestrian movement in building spaces, demonstrated in bottleneck scenarios and crowd merging. As the model provides a realistic throughput model, it can be used to determine the maximum throughput of groups on edges in the building network flow graph. As part of the model investigation in Chapter 3, we showed the phenomenon of a "peak throughput" in simple corridor bottlenecks, the point after which increasing the supply of agents causes a reduction in throughput. By discovering the appropriate peak throughput value for each edge in the graph, we can then establish these values as the "Capacity" parameters for the edges. This approach allows the (macroscopic) graph to accurately reflect the flow performance of the (microscopic) simulation, termed Micro-Macro Coupling.

To accomplish this, we iterate through each edge in the graph to determine the maximum throughput. This is determined by attempting a low initial supply rate of agents at the start of the edge and instructing them to travel to the end of the edge, where they are removed from the simulation and their travel time noted (Figure 42). The supply of agents is gradually increased over time until the rate of supply exceeds the rate of removal by a threshold of 5% (to avoid variance in agent travel to cause premature cessation of coupling). At this point, the throughput capacity has been

identified (Capacity) and is recorded for the edge, along with the average traversal time for the low initial supply (giving the traversal time for low-congestion on the edge: mintrav) and the average traversal time at the maximum supply (giving the traversal time for a fully-utilized edge: maxtrav), Algorithm 7.

Algorithm 7: Flow Coupling

```
Data: PlannerGraph pg
foreach Edge e ∈ pg do
    Node a = e.firstNode;
    Node b = e.secondNode;
    attempted = 0.01;
    achieved = 0.01;
    maxtrav = 0;
    mintrav = 999999;
    while attempted/achieved ≥ 0.95 do
        escaped = 0;
        for 1000 simulation ticks do
            add +=attempted;
            if add ≥ 1 then
                add a new Agent at a, moving to b;
                add–;
            end
            foreach Agent P at b do
                escaped++;
                if P.traversalTime > maxtrav then
                    maxtrav = P.traversalTime;
                end
                if P.traversalTime < mintrav then
                    mintrav = P.traversalTime;
                end
                destroy(a);
            end
        end
        achieved = escaped/1000;
        attempted+= 0.01;
    end
    e.capacity = achieved;
    e.maxtrav = maxtrav;
    e.mintrav = mintrav;
end
```

Edge coupling is an automated process which takes some time; EvacSim needs to simulate each attempted supply for a long enough period of time to achieve a stable average throughput achieved (we use 1000 ticks for each attempted supply) and as each edge takes many different attempted supplies in order to converge on the optimum achieved throughput this can take several hours (140 minutes for the "B2" building from Chapter 4).

In this work we simply increment the attempted supply by 0.01 occupants-per-tick, each 1000 ticks; some time improvement could be achieved by using a half-interval approach to converge on the optimum throughput more quickly (increasing the throughput attempted by larger amounts early in the coupling of an edge to converge more quickly on the maximum throughput).

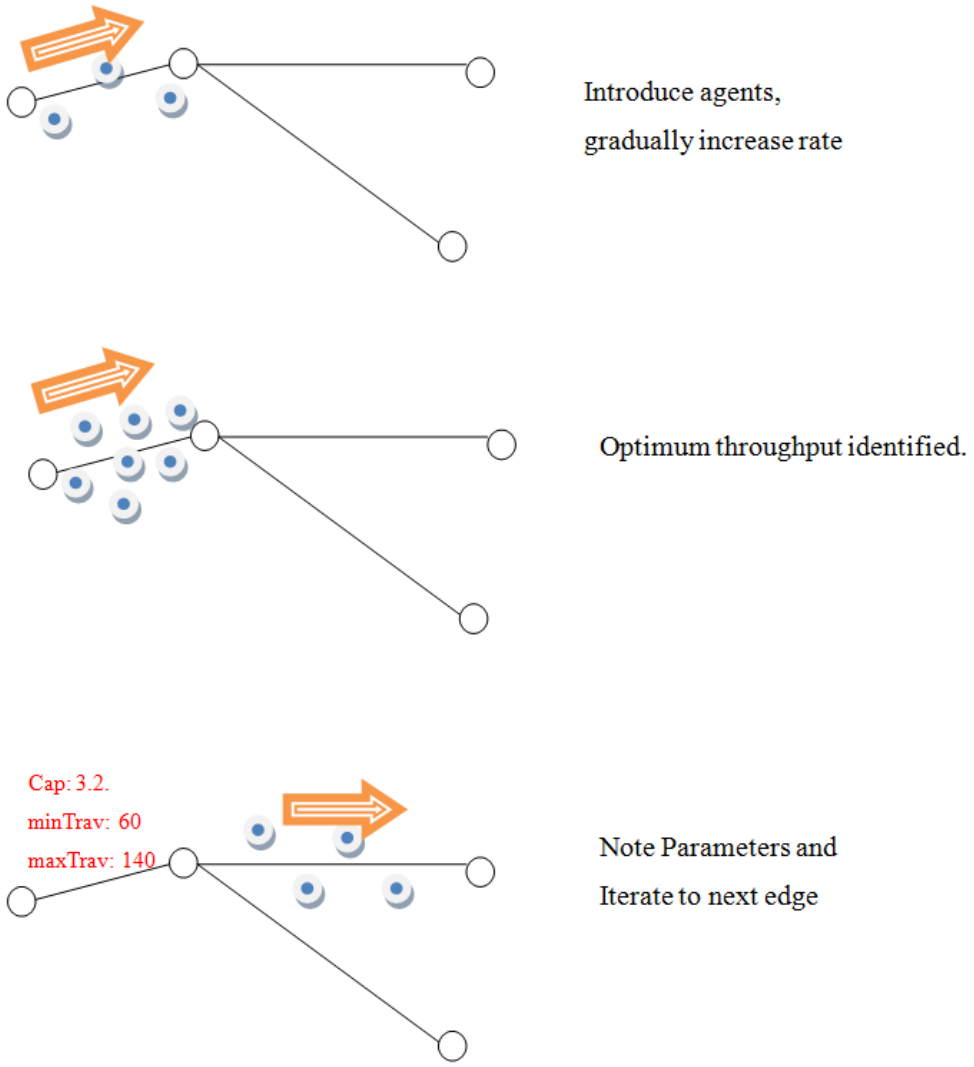


Figure 42: Path Length Accuracy (Coupling Edges)

The coupling proceeds for each edge in the graph until all edges have been tested, at which point the graph has been fully coupled and can be used for dynamic evacuation planning, or evacuation outcome prediction. As the flow characteristics in one direction may differ from those in the reverse, each edge in the graph is considered individually as a unidirectional edge (with the equivalent reverse direction coupled separately.

5.4 Path Testing in buildings

To determine the effectiveness of the micro/macro coupling approach in network flow graphs using EvacSim, we performed experiments moving groups of agents along multi-edge paths in the building. If the coupling is accurate, then the throughput of the path should be approximately the same as that of the edge with the lowest throughput value. Similarly, the average traversal time for agents in the experiments should fall between the low-congestion time and the maximum throughput traversal time values.

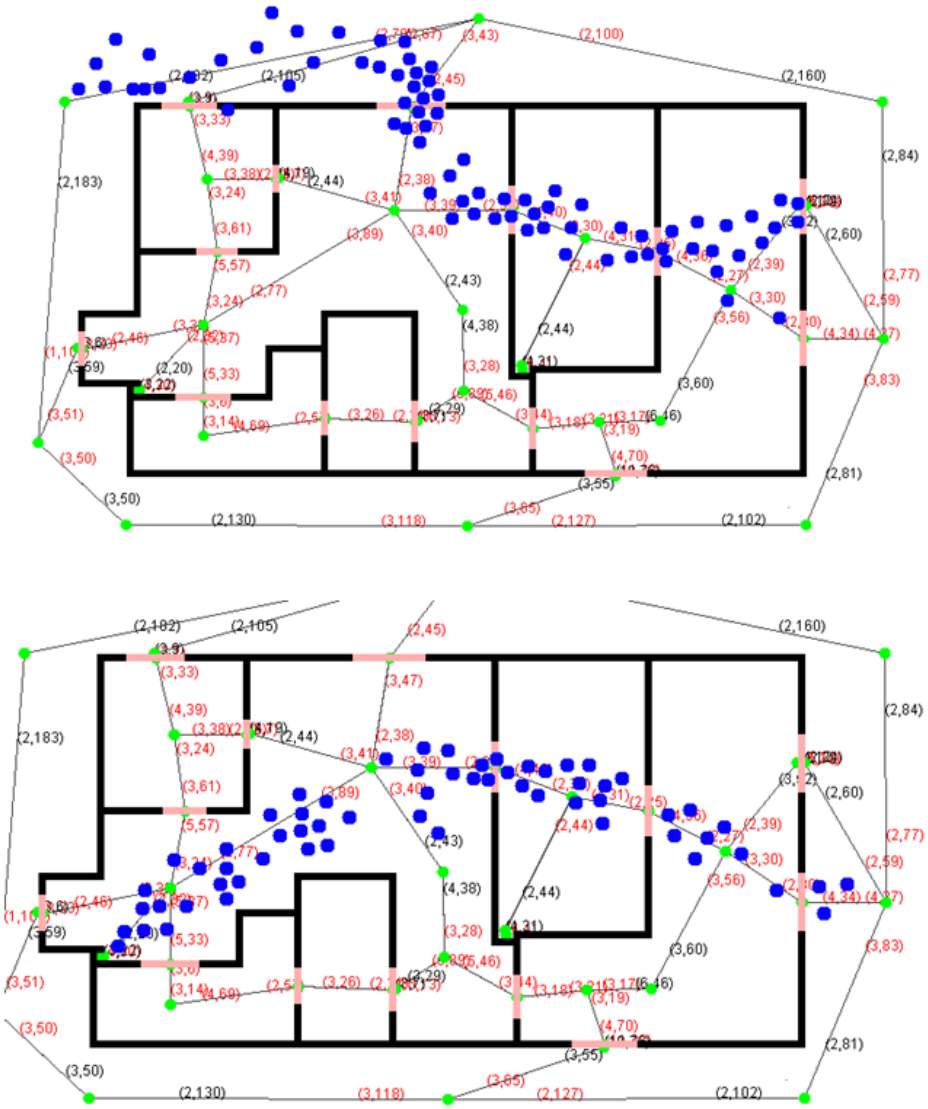


Figure 43: EvacSim screenshots of two examples of path testing in progress

These experiments were performed on the B2 building (and associated coupled graph, Figure 43) used in Chapter 4. The numbers associated with either end of the edges on this graph indicate the capacity (rounded to nearest integer) and average of minTraversal and maxTraversal (again, rounded). The tuple near a node indicates the flow parameters when departing that node.

By introducing a supply of agents at a randomly selected node of the graph and instructing them to travel to another randomly selected node using a path chosen by using the A* shortest path algorithm [8], we can then observe how closely the achieved throughput corresponds to the expected throughput of the path. To determine this, the initial supply is set to half the capacity of whichever edge in the path has lowest capacity. This supply is then periodically incremented by small amounts, as in the initial per-edge coupling procedure in Section 5, until the achieved throughput peaks. The accuracy of the graph is measured in terms of the ratio between throughput achieved and throughput expected (Figure 44) and the real difference between the achieved capacity and expected capacity (Figure 45).

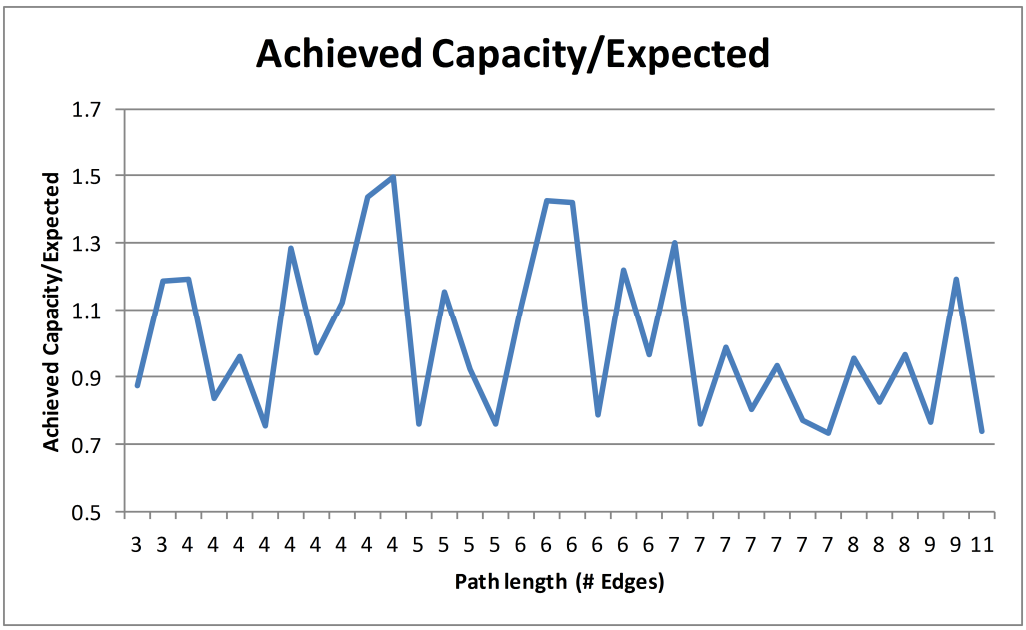


Figure 44: Path testing: Capacity estimate accuracy

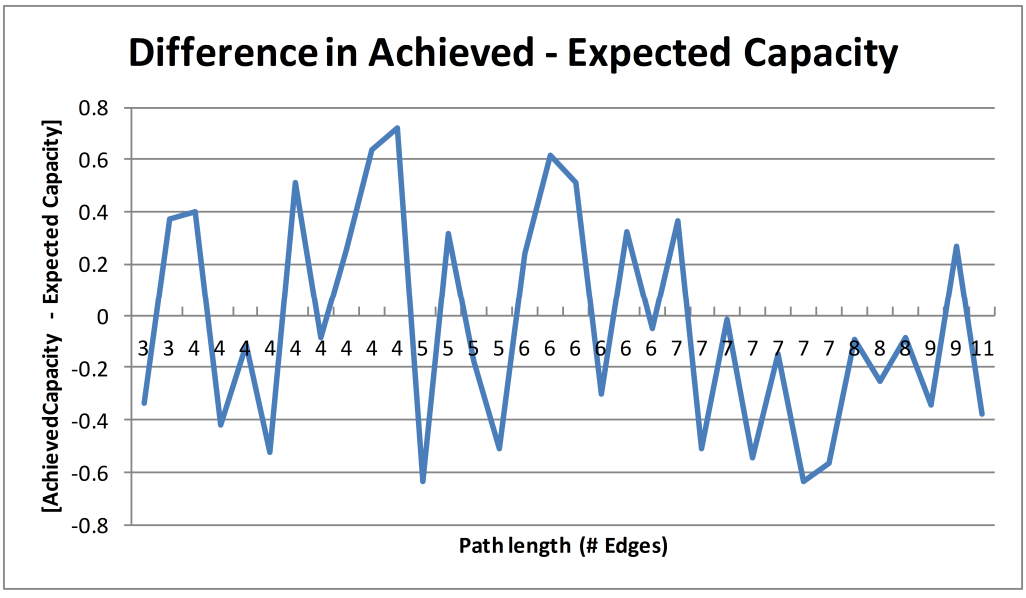
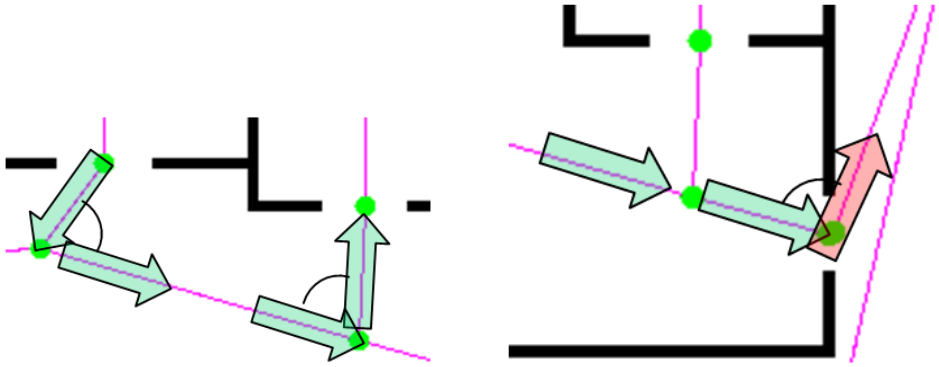


Figure 45: Path testing: Capacity Real Difference

In these experiments, we found that the error has a similar tendency in either di-

rection, small over- and under-estimations over the course of a path tend to balance out. Error is often a result of a change in angle during the path, where the occupants slightly cut the corner. Identifying these angle changes on the graph led to no conclusive association between the angle change and over- or under-estimation (Figure 47), where we see the rate of over- and underestimation is unrelated to the number of high-degree turns.

The error is a result of the combination of angle change (allowing for a corner to be cut, but also compressing the crowd) and building geometry occluding a potential shortcut, impeding the flow (Figure 46). This result reinforces the assertion that while Macroscopic Flow Graphs perform well for modelling pedestrian movement, detailed microscopic simulation captures more detail of occupants freely navigating physical spaces.



(a) Unimpeded turns (b) Turn impeded by geometry

Figure 46: Path angle changes and error (geometry occlusion)

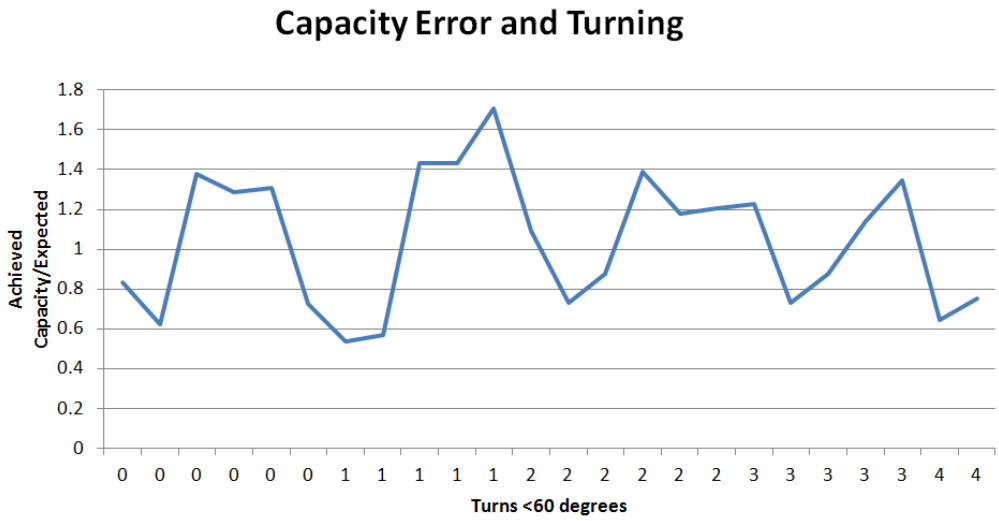


Figure 47: Error and Turning (change in angle)

In these results, we observe no particular pattern relating the length of paths to the accuracy of their capacity estimates other than that longer paths tend to slightly

overestimate the capacity as can be seen in paths of 7+ edges. Overall the estimates are quite close to the achieved values, with the average ratio between Achieved and Estimate being ∼1.01:1 in these experiments, with a standard deviation of ∼0.24 (or ± ∼24%).

Traversal times were similarly accurate (Figure 48), with an average ratio (over 34 trials) of Achieved to Estimate of ∼1.05 and a standard deviation of ∼0.2. The real difference between Expected Capacity and Achieved Capacity (Figure 49) fell within a fairly narrow range, averaging ± ∼0.03 with a standard deviation of ∼0.4. Again, the traversal time estimates are have a small real amount difference to the traversal times achieved, giving an average real difference of 8.8 time units to traverse compared to that achieved during path tests, with standard deviation of ∼28 for these results.

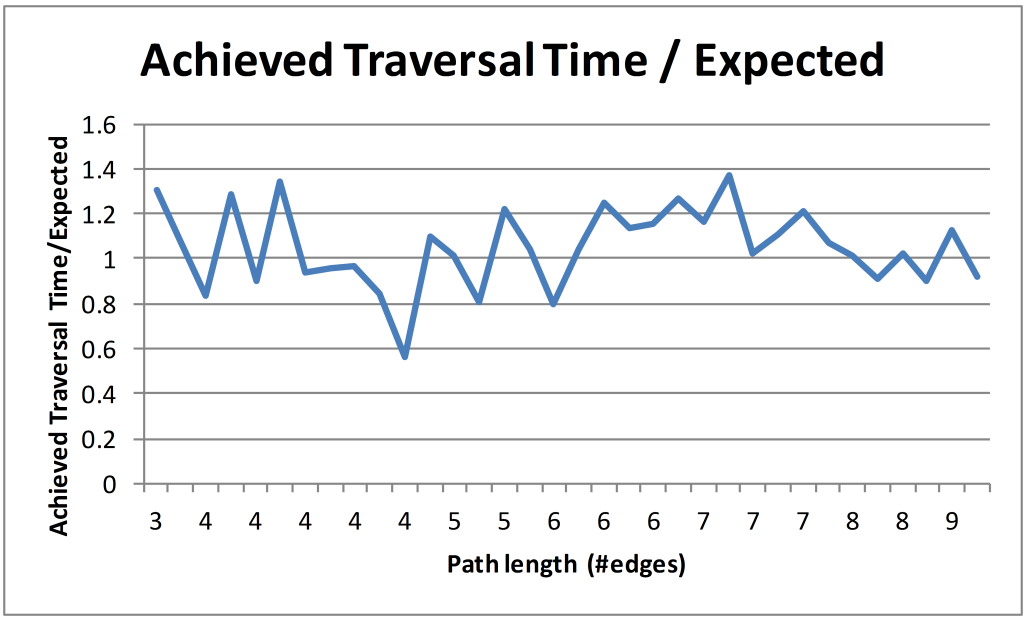


Figure 48: Path Testing: Traversal Time Accuracy

These results demonstrate that while Macroscopic Flow Graphs feature a much lower level of detail and complexity relative to microscopic multi-agent simulation, through coupling they can be configured to accurately reflect the movement characteristics of groups in the micro simulation. While the graphs have a much lower level of fidelity compared to micro simulation, through micro-Macro Coupling we can provide them with realistic capacity and traversal time values which correspond closely to the results achieved using microscopic agents in multi-edge paths. In Section 5.5 we will demonstrate the utility of these coupled graphs for prediction of arrival times for groups of occupant agents travelling between arbitrary nodes in the building.

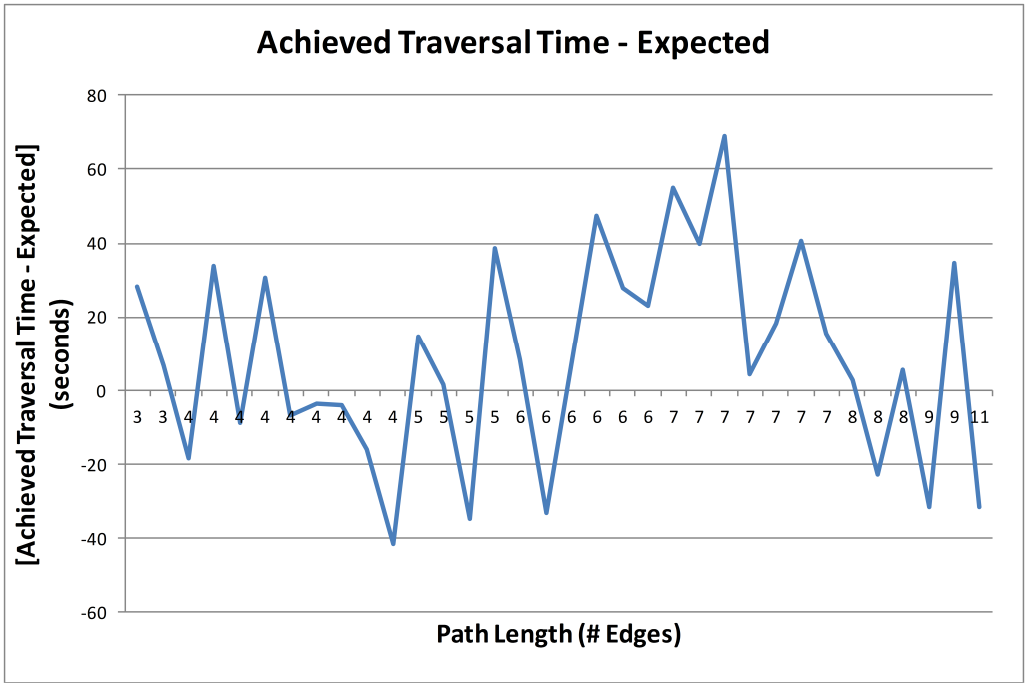


Figure 49: Path testing: Traversal Time Real Difference

5.5 Traversal Time Prediction

While the estimated overall capacity and traversal times for persistent, maintained flows of occupants matched closely with the achieved results, it is important to be able to make accurate prediction for limited flows of occupants. That is to say, predicting the first-and-last arrival times for a limited group of occupants travelling from one part of the building to another would be a key benefit of graph-based analysis. By predicting the arrival times in this manner, we could attach a probability to where occupants are likely to be in the near future without simulating it first. Such predictions could be used for tasks such as problem-space-decomposition (dividing the simulation world into discrete sub-simulation) if it can be determined that different groups are not likely to meet at the same place at the same time (and hence interfere with one another, which would require micro simulation to model accurately).

Unlike the constant flow scenarios in the path testing of Section 5.4, loss in traversal time caused at congested edges can be made up by the group on other, less constricted edges. With micro-Macro coupled graphs, we can estimate the time taken for a group of occupants to traverse a given path by taking the summed average minTraversal and maxTraversal values of each edge in the path, and adding on the delay caused by queuing due to limited capacity. Experiments using limited crowd sizes rather than the steady flow of occupants used in path testing suggest that the average traversal time for individual agents falls between the total minTraversal and maxTraversal values for the path. In these experiments, a group of occupants of random size (between 5 and 25) is generated in EvacSim at a random node loca-

Table 3: Group Traversal time estimate accuracy, first arrivals

Achieved:estimate	0.93:1
Standard Deviation	0.13:1

tion and instructed to travel to another random node in the graph, with the path computed using A* (Figure 50).

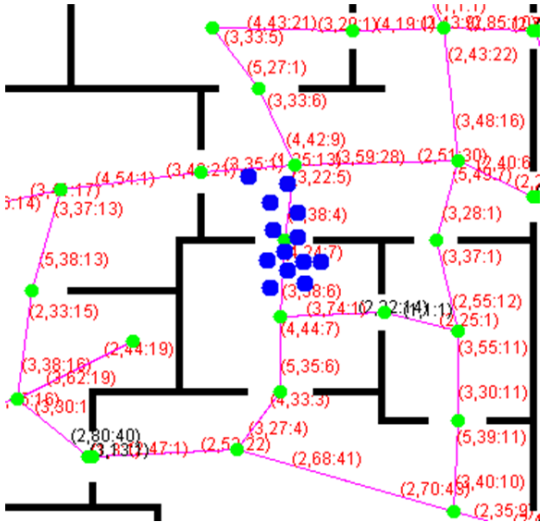


Figure 50: Group Traversal Time Experiment screenshot

The time taken for the first agent to arrive at the destination is then compared with the average of minimum and maximum traversal time estimates from the graph: having identified the edges on the route the group is taking, we can compute the estimated first arrival time using (2). with the ratio between this estimate and what was measured shown in Table 3.

$$\frac{\sum_{i=0}^{n}(MaxTraversal_i) + \sum_{i=0}^{n}(MinTraversal_i)}{2} \tag{2}$$

As this approach shows good accuracy for first arrival prediction, we next need to predict the time taken for the last occupant in the group to arrive at the destination. With these two values, we have the bounding times where the crowd is present at a given node. To estimate the last arrival time, we account for the delay caused by queuing at the most congested edge (the edge with the smallest capacity value) and add this to the time taken for first arrival, 3.

Table 4: Group Traversal time estimate accuracy, last arrivals

Achieved:estimate	0.98:1
Standard Deviation	0.17:1

$$\left(\left(\frac{\sum_{i=0}^{n}(MaxTraversal_i)+\sum_{i=0}^{n}(MinTraversal_i)}{2}\right) + \left(\frac{crowdsize}{minimumCapacity([0-n])}\right)\right) * TimeUnitAlpha \tag{3}$$

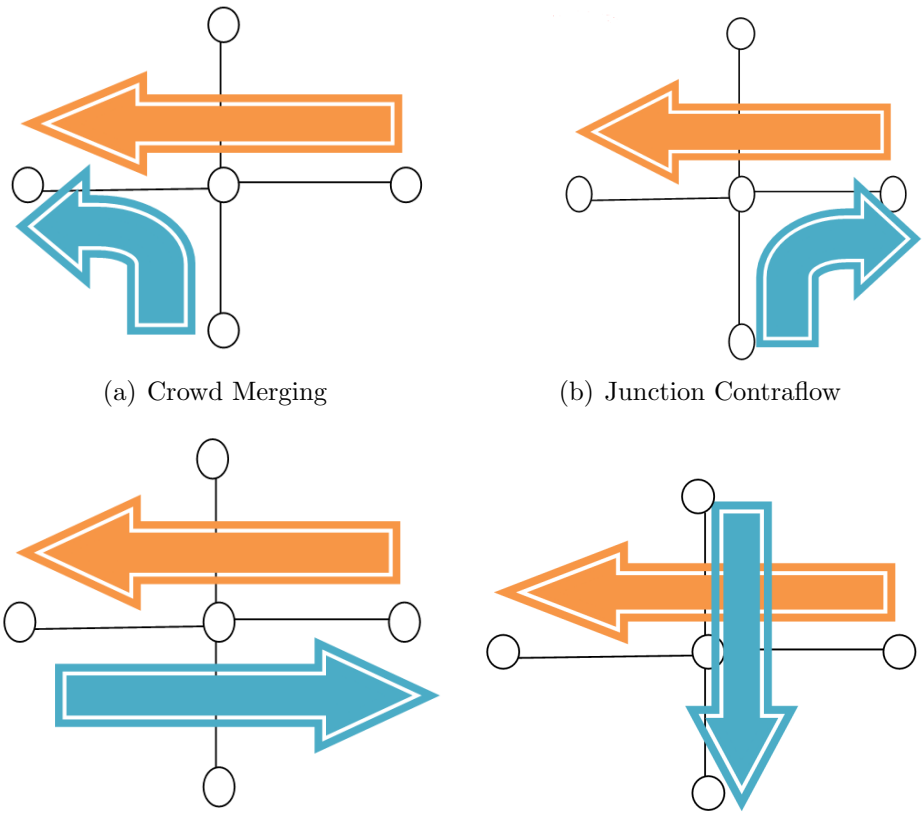
In Table 4 we show the averaged ratio of achieved to estimated last arrival times (estimates computed using 3). In these result we see that the ratio is quite close to 1:1.

These results demonstrate that by using micro-macro coupled graphs, we can make accurate predictions about the time taken for crowds to traverse between arbitrary points in a building, by not only accounting for the traversal times of the edges (based on the average of minimum and maximum traversal time values) but also by accounting for the size of the crowd and the capacity of edges. By coupling the macroscopic graph to the microscopic simulation to produce accurate capacity and traversal values, we can use the macroscopic graph as a powerful predictive tool in its own right.

5.6 Flow Graph Limitations

In this chapter we showed that coupled Flow Graphs can be useful for predicting traversal times and congestion for simulated occupants in microscopic simulation. While this is certainly true in the case of single flows of agents (Section 5.4) and single crowds of occupants (Section 5.5) there are limitations to the Flow Graph approach which become apparent when simulating multiple simultaneous flows with the EvacSim microscopic model. In particular we note that Flow Graphs do not model contra-flow scenarios well. With bidirectional edges, examining Flow Graphs alone suggest that a maximum flow is achievable simultaneously in both directions, which is extremely unlikely. Similar difficulties arise when considering simultaneous arrival of two separate streams of occupants at a single common node, or attempts to split flow in two directions departing an origin node. A related problem is that of two streams crossing each other at a common node (a "crossroad problem"); while no edge appears on both paths, the collision of both streams results in much lower flow than suggested by the graph. In this section, we investigate the impact

of these graph limitations on traversal time prediction, by testing simple examples on a subsection of the B2 building graph which features a roughly cross-shaped section. The four scenarios tested are Crowd Merging, Junction Contraflow, Direct Contraflow and Crossroad (Figure 51).



(a) Crowd Merging

(b) Junction Contraflow

(c) Direct Contraflow

(d) Crossroad

Figure 51: Simultaneous Flow through junction experiment scenarios

We point out that in each of the heatmaps for this section, the area around the starting positions for crowds tends to have cool blue heatmap points as the agents begin the experiment in stationary positions and hence appear to have low velocities in these early heatmap points.

5.6.1 Crowd Merging

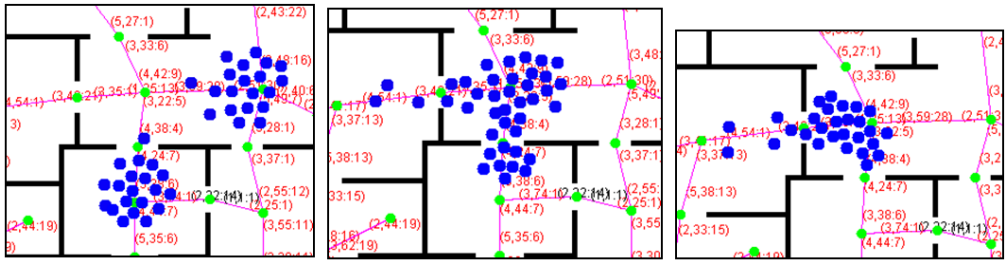


Figure 52: EvacSim Screenshots: Crowd Merge Experiment (start, middle, end)

Table 5: Crowd Merging traversal time prediction results

First Arrival:Estimate	1.01:1
Standard Deviation	0.14:1
Last Arrival:Estimate	1.06:1
Standard Deviation	0.06:1

By repeating the experiments in 5.5, but using two separate crowds in the simulation, we can demonstrate the limitation of using flow-graphs alone. To show the impact of merging two crowds that have different starting positions but common destinations (Figure 51(a)); this experiment is repeated in EvacSim 10 times with random crowd sizes (Figure 52) . While the first-arrival time for this scenario using Formula 2 holds, the last arrival time (Formula 3) underestimates the amount of time taken by a small amount.

These results (Table 5) show that the merging of the groups causes a reduction in the orderly movement of the crowd that causes slower overall traversal compared to a homogenous crowd all traveling in the same direction, a result reflecting the impact of crowd merges identified in Chapter 3. The difference between the merging area and the post-merge can be seen in the velocity heatmap image (Figure 53) where the area of the merge is cooler, with warmer areas after merging as agents reform a stream and have passed through the narrow doorway.

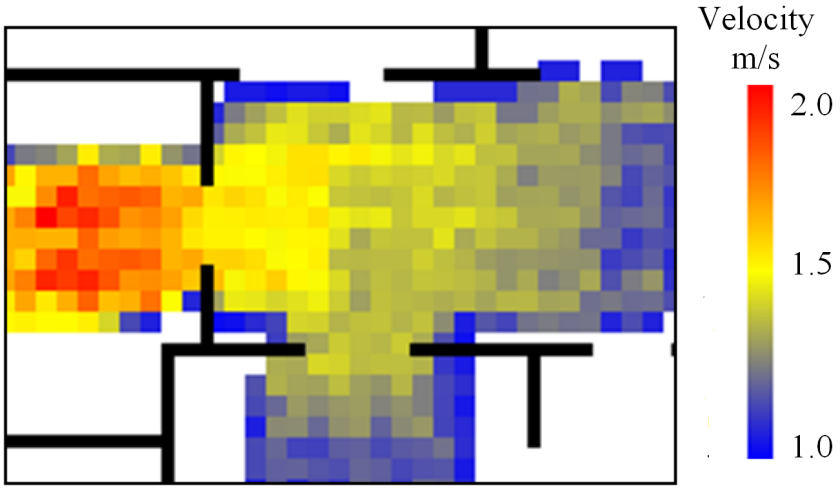


Figure 53: Crowd Merge Experiment (Velocity Heatmap, EvacSim screenshot)

5.6.2 Junction Contraflow

The next scenario investigated is that of "Junction Contraflow" (Figure 51(b)), where one group travels a straight path, and the other arrives perpendicularly onto the first group and attempts to travel to the first group's starting position.

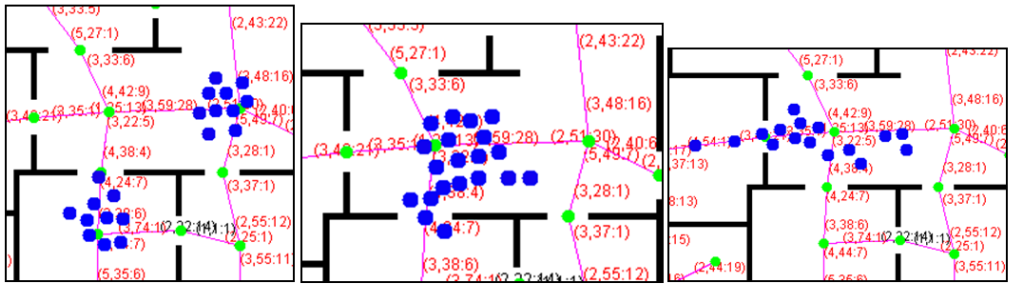


Figure 54: EvacSim Screenshots: Junction Contraflow Experiment (start, middle, end)

Table 6: Junction Contraflow traversal time prediction results

First Arrival:Estimate	1:1
Standard Deviation	0.21:1
Last Arrival:Estimate	1.57:1
Standard Deviation	0.09:1

As with the Crowd Merging experiment, crowds are generated in EvacSim and instructed to traverse from their start to finish nodes (Figure 54). Having performed these experiments, we can see the impact of the crowds interaction in the velocity heatmap (Figure 55). The area around the common node features slower movement as agents jostle to pass through the junction and while the first arrival estimates remain accurate, the last arrival times are approximately 50% greater than the estimates (Table 6) as occupants get delayed in dealing with the two counteracting flows.

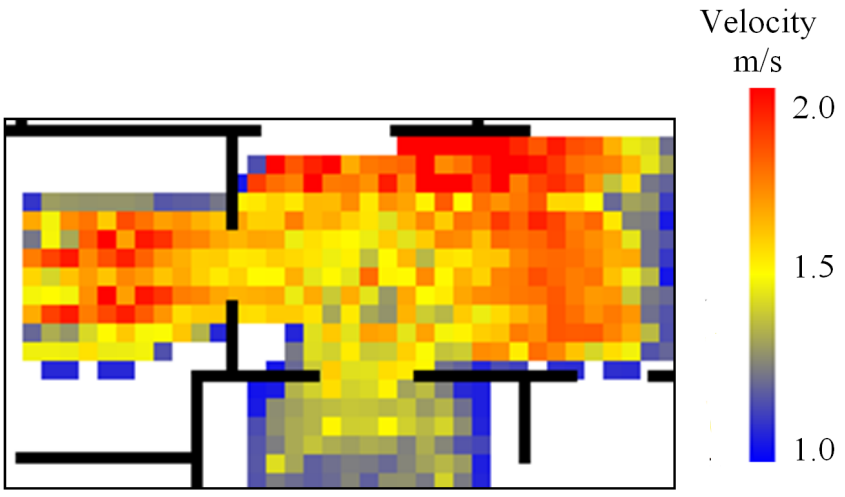


Figure 55: Junction Contraflow Experiment (Velocity Heatmap)

5.6.3 Direct Contraflow

In the next experiment we show the impact of direct contraflow in heavily subscribed edges. In this experiment, the crowds start at opposite ends of the same path and

traverse to the other side (Figure 51(c)). The collision of the crowds causes a great deal of congestion and difficulty, with the attempted contraflow at the doorway node greatly increasing the time taken for the two crowds to complete their paths.

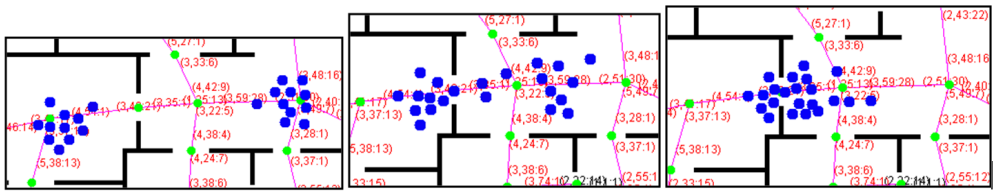


Figure 56: EvacSim Screenshots: Direct Contraflow Experiment (start, middle, end)

In these expriments (Figure 56), we find that the ratio of First Arrival:Estimate are still close to 1:1, as one or two agents from each group manages to squeeze past the other group at an early stage; however the last arrivals take substantially longer than suggested by Formula 3, as jams form at the area of most limited capacity and the two flows are in direct opposition (Table 7). In these experiments, the last arrival tends to take two-and-a-half times as long as the estimate, though the Standard Deviation for last arrival is quite high, which relates to the size of the crowds. Two small groups can quickly clear the bottleneck before the congestion effects become more serious, relative to two large groups which result in protracted jams. The narrow space where the jam forms is clearly visible on the velocity heatmap (Figure 57) as the cool area located around the narrow doorway.

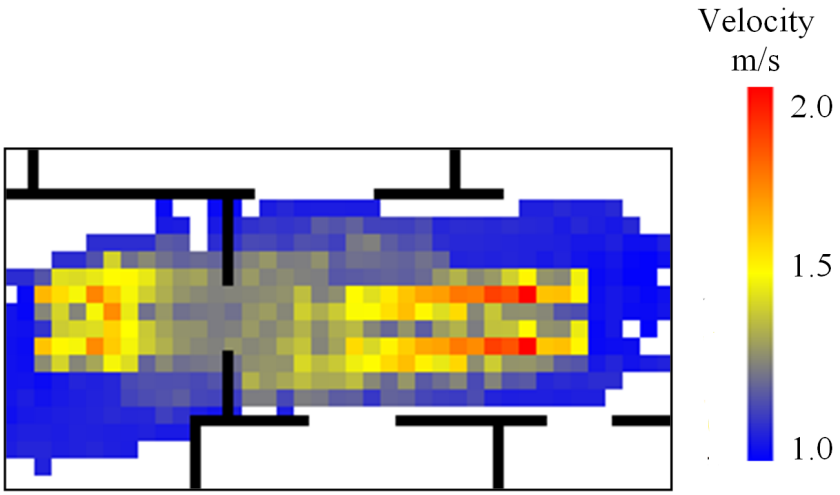


Figure 57: Direct Contraflow Experiment (Velocity Heatmap)

5.6.4 Crossroad Junction

In this final experiment, we show the impact of two streams crossing in an X-shape: a crossroad configuration (Figure 51(d)). In this experiment, collisions form at the common central node as the two streams attempt to navigate the area. As there is

Table 7: Direct Contraflow traversal time prediction results

First Arrival:Estimate	0.84:1
Standard Deviation	0.07:1
Last Arrival:Estimate	2.48:1
Standard Deviation	0.09:1

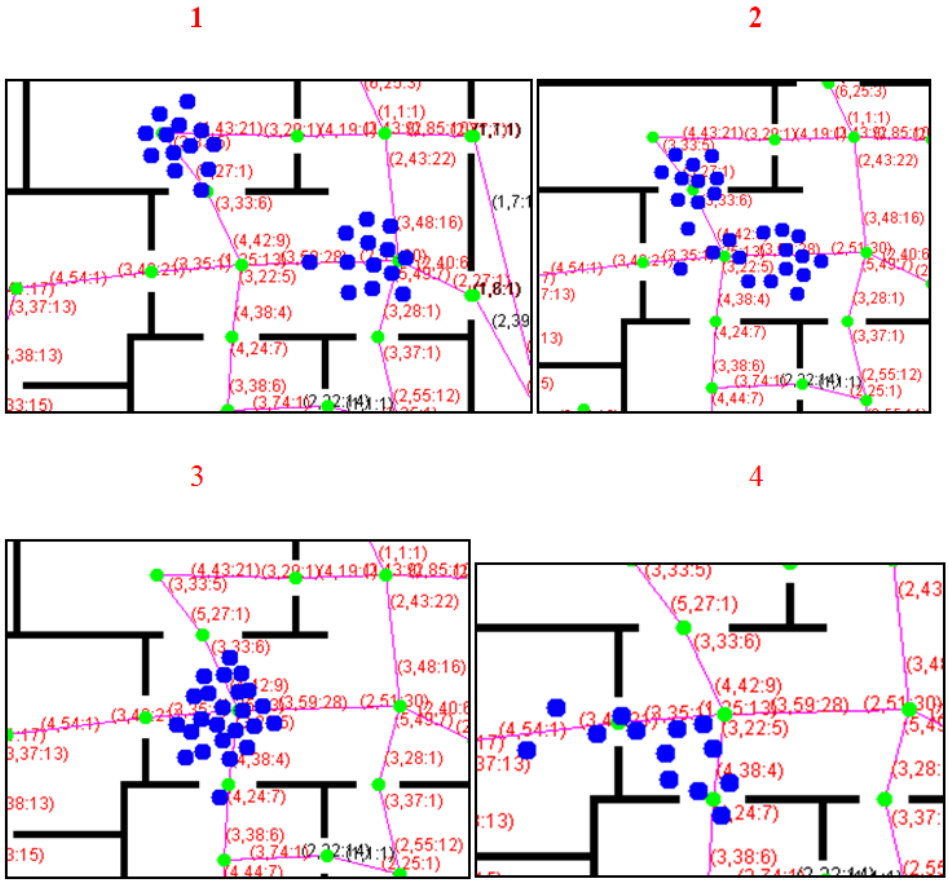


Figure 58: EvacSim Screenshots: Crossroad Experiment (start, early, late, end)

not a significant congestion bottleneck here, the crowds can make use of the space in the area to spread out and negotiate the collision (Figure 58). The results of this experiment (Table 8) show that the first arrival times are slightly slower than the estimate, as the collision in the central node causes the agents to have to slow down and negotiate the area.

As with Direct Contraflow (Section 5.6.3), the crossroad experiment showed an underestimation of last arrival by a factor of 2.5, however in this case the variance was slightly lower as the overall congestion of the common space is lower compared to that going through the narrow door in direct contraflow, and some degree of allowance is afforded to the agents in the common space as seen in the yellow central area in the velocity heatmap (Figure 59). It can also be observed in the heatmap that once the collision area is passed, the flow is orderly and picks up speed as

Table 8: Crossroad Junction traversal time prediction results

First Arrival:Estimate	1.22:1
Standard Deviation	0.27:1
Last Arrival:Estimate	2.58:1
Standard Deviation	0.68:1

observed by the red bands after each doorway.

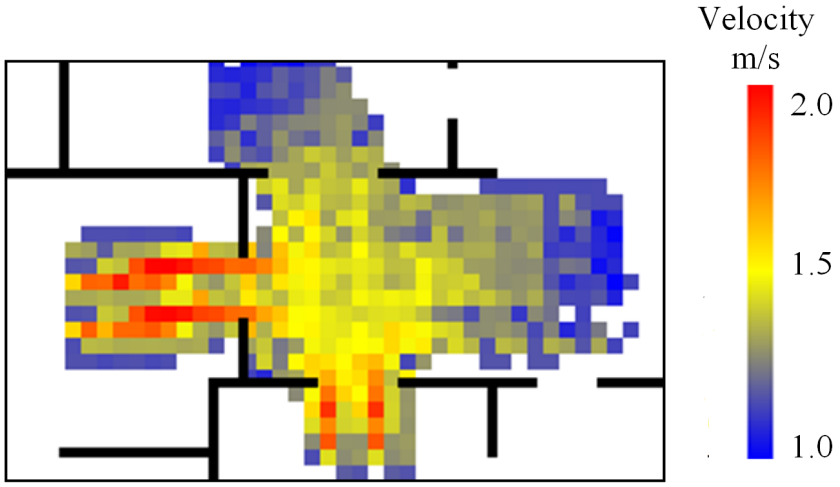


Figure 59: Crossroad Experiment (Velocity Heatmap)

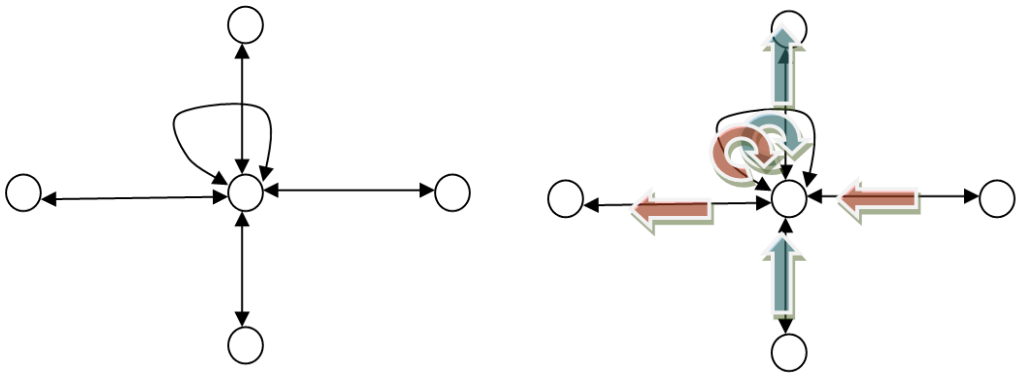
5.6.5 Possible Approaches for Overcoming Flow Graph Limitations

While Flow Graphs performed well for traversal time predictions for single-group problems, the limitations of this approach in handling multiple groups traversing a common space became apparent when testing junction and contraflow situations. When modelled as seperate, unidirectional edges, the path between two points overestimates the achievable throughput. Without modification, the graph model considers two contradicting flows to be independent when in reality they are both present in the same place and directly interacting in the microscopic model. Modifications to standard flow graph models could alleviate this issue; by associating each directed edge with its reversed equivalent, simultaneous capacity constraints could be modelled on the graph. Traffic on a single edge could use the full capacity of the edges, but two groups traversing in opposite directions could only use a portion of the *simultaneous* capacity of the two edges.

In addition to considering the simultaneous capacity of opposing edges, graphs models could model the simultaneous arrival and departure at nodes as a seperate capacity in addition to the edge capacities. In the Junction ContraFlow and Junction Crossroad experiments, agents were impeding each other when arriving and departing a common node, despite being on seperate edges. When multiple flows are

passing near each other at common nodes, they interact and the total achieved flow is lower than the sum of the individual edge flow capacities. Considering the simultaneous arrival and departure of flows as an additional constraint on traffic capacity could improve the graph model; discovery of the arrival/departure capacity for multiple edges at a common node could be achieved through a special case of micro-macro flow coupling, using multiple attempted flows.

In each of these junction experiments, the flows pass through a central junction node. A simple approach to capturing the phenomena of multiple flows traversing common space would be to augment nodes to model capacity, in addition to the edge capacities currently used. This could be achieved by assigning capacity values to the nodes themselves, or to generate small "loop" edges on each node, which must be traversed first before flows move onto the departing edge. The limited capacity of the loop edge would provide a simple means of modelling the simultaneous use of space when different flows pass through a common node (Figure 60).



(a) Loop Edge added to junction node

(b) Simultaneous Flow can be detected on the Loop Edge

Figure 60: Loop Edge for detecting interaction of flows at junction

5.7 Simulated Evacuation using a Flow-Based Planner and coupled graphs

5.7.1 Introduction

Having shown that coupling of these graphs to the microscopic EvacSim simulation led to accurate correspondence between the macroscopic graph perspective and the simulation, we can now make use of the flow graph as input for a Flow-based Dynamic Evacuation Planner. The planner used in this work is "EvacPlan", a Java implementation of Hadzic's dynamic flow-based evacuation planning algorithm [33], which takes as input the current occupancy state of the building, along with hazard locations and a network flow graph model of the building space. Using these,

it computes evacuation routes for building occupants in emergencies by reasoning about the size of groups, the traversal time required to navigate edges, and the limited capacity of edges. For dynamic evacuation planners to provide reasonable and safe plans, the flow parameters need to accurately reflect the reality of the building space.

In these sections we investigate the performance of EvacPlan in evacuation scenarios using the coupled flow graphs generated in Section 5.3. Evacuations are performed on a building with its coupled graph, varying the population of occupants and using EvacPlan plans with a variety of allocation strategies used for distributing these plans among agents (allocations based on occupant position, sharing of paths and using initial wait times before beginning evacuation). These results are then compared with naïve evacuation strategies based on shortest-path evacuation, and safety-based plans that maximize distance from danger. By investigating EvacPlan's performance in EvacSim, we exploit the micro-macro coupled graphs from Section 5.3 for dynamic evacuation planning, and also explore the problem of allocating multiple seperate macroscopic-graph-based evacuation routes to populations of macroscopic agents, and how different allocation approaches impact on evacuation times and safety.

5.8 EvacPlan Dynamic Evacuation Planner

EvacPlan is a Java implementation of Hadzic's planner [33], programmed by Dr. Tarik Hadzic and used as a module in EvacSim. It takes as input:

- Flow Graph Nodes
- Flow Graph Edges (with Capacity and Traversal Time)
- Hazard starting locations (Graph Nodes)
- Building Exits (Graph Nodes)
- Building Occupancy (Number of occupants at each node).

EvacPlan computes evacuation routes by considering these elements and using a multiple heuristic-based approach that combines safety heuristics (avoiding hazard) while minimizing evacuation time (shortest distance), while considering the limited capacity of graph edges.

5.8.1 Node Occupancy

As occupancy state is given in terms of an occupancy count on each node in the building, before invoking EvacPlan, EvacSim performs the "whereIsThis" (Algorithm 17

call on each agent in the simulation, which returns the appropriate Space and associated Graph Node for that agents. From this information, the number of agents at each node location is registered and established as the current "occupancy" of that node. Alternatively, simulated sensors could be associated with building nodes and the sensed data used to determine occupancy (or indeed, real sensors in a real-world evacuation) but this approach was not used for these experiments.

5.8.2 Edge Occupancy

EvacPlan does not model capacity at nodes, but rather limits the capacity on edges. When EvacPlan is invoked, it produces a set of plans for each node, with a count for each plan indicating how many occupants should use each particular plan. Also attached to these plans is a "delay" value, which indicates how long occupants should wait before beginning their route. As such, EvacPlan assumes that once an occupant begins their route, they can continue at the average occupant walking speed all the way to the exit without being impeded by congestion effects. A consequence of this assumption for EvacPlan models is that while agents start at a node position, once they begin their path they will never stop at another node (and thus, EvacPlan considers occupants to be present on edges rather than nodes once their route starts).

5.8.3 Hazard Locations

Hazard starting location is given as a Node in the graph, and EvacPlan produces a simple internal hazard spreading model based on the spreading of hazard along graph edges. The hazard spreads between neighbour nodes at a rate given by 4:

$$\frac{HazardSpreadRate * TimeUnitAlpha}{maxTraversalTime} \tag{4}$$

This spreading model is used by EvacPlan to avoid plans that cause exposure to the hazard. EvacSim tracks hazard location by polling EvacPlan to request the set of nodes that are currently "In Hazard", with the associated building spaces filled in with a Fire icon in the EvacSim display. In the evaluation experiments, any agents present at "In Hazard" locations in EvacSim are considered injured and no longer participate in evacuation. While the EvacPlan hazard spread model is somewhat crude and unrealistic, in this work we are exploring the interfacing of macroscopic graph planning models with microscopic agent movement models to evaluate how well the EvacPlan plans perform in the more detailed microscopic environment, and the impact of different path allocation strategies on evacuation outcomes. In real-world deployment of simulation for evacuation more detailed hazard models would

provide for a richer simulation overall; however as we are evaluating pedestrian movement models rather than fire spread simulation, we use the EvacPlan hazard spread model as the model for hazard spread in EvacSim.

5.8.4 Exit Locations

EvacPlan models the Exits of the building in the form of a single "Exit" node, with edges of infinite capacity and 0 traversal time connecting this node to the building's emergency exit locations. In these experiments, we select perimeter doorway nodes as emergency exits and Planner Evaluation Experiments. To investigate the performance of the EvacPlan evacuation planner using microscopic simulation, we set up emergency evacuation scenarios using the grid-like B2 building from Chapter 4, with 4 exit nodes, placed at the top, sides and bottom of the building (Figure 61). This building model was chosen for these experiments as it features a variety of topology features which will provide numerous challenges for the planner; some perimeter doorways are close to each other, and there are many junctions where different flows might interact.

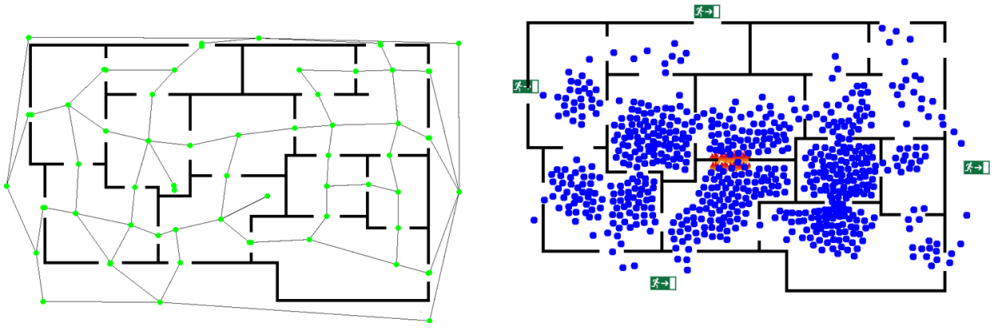


Figure 61: EvacPlan evaluation building ("B2")

5.8.5 Experiment setup

In these experiments, we begin with an initial deployment of occupant agents, scattered throughout the building at randomly generated positions. The agent node locations (given by "whereIsThis", Algorithm 17) form the basis of the Node Occupancy data given to EvacPlan for computing routes. At the beginning of each experiment, a hazard location is chosen (placed at the centre of the building in these experiments), which are converted to a graph node via "whereIsThis") and EvacPlan computes evacuation routes for the agents in the building, which are then allocated to the agents and enacted by the agents using the Hybrid Path Planning (Section 4.9).

Experiments are conducted varying the overall population from 100 agents to 900. Agents have parameters established based on Chapter 3, with walking speeds taken

from a standard distribution of typical pedestrian speeds as validated in that Chapter. Agent starting positions and speed distribution are chosen using the same random seed for each experiment instance, to allow for experiment reproducibility and fair comparison between strategies. Evacuation performance is measured based on two values:

- the amount of time (in simulation ticks) taken for 95% of the uninjured building population to successfully clear the building
- the number of injured agents (the number agents exposed to hazard for more than 10 ticks before 95% evacuation completion)

5.8.6 Plan Allocation Strategies and Building Setup

These experiments are performed using several path allocation strategies. As EvacPlan produces routes that are allocated to occupants of the building, we approach dynamic plan allocation using three different strategies:

- Arbitrary Plan allocation (arb)
- Proximity Allocation Plan Allocation (prox)
- Proximity Allocation Plan Allocation with Delays (prox-d)

The first of these provides individual routes to each occupant, with the plans allocated arbitrarily to agents at each node without consideration for their specific position beyond being present at the first node of a path. The second strategy (*prox*, assumes a finer degree of localization and communication in which it is possible to determine which occupants are nearest to the first step of each plan, and thereby allocate the plans on this basis (resulting in an orderly distribution of routes and avoiding agents on different routes coming towards each-other at the early stage of the evacuation. This strategy assumes a fine degree of localization and communication control that may not be realistic in real-world scenarios, but will result in flow that more closely matches what EvacPlan would expect, at least early on in the evacuation.

The third approach allocates the paths using delay values (path delays are used by EvacPlan to request that the occupants do not start on their allocated path until after a given delay period). In this manner, we can achieve a more orderly start to evacuation with a controlled rate of departure, and combined with the Proximity Allocation strategy, conforms the most closely with EvacPlan's expectations, though assumes that the occupants receiving the paths are willing to obey the delay request. For comparison purposes, we also compare the performance of these path allocation strategies with two alternative evacuation plan generation options provided by the

EvacPlan module; these options simply focus on one heuristic (safety or time) at the expense of the others:

- Shortest Path (shortest path in traversal time to any exit)
- Safest Path (a path that avoids edges that the hazard might spread to soon)

For further comparison, investigating scenarios where communication or control is limited, the EvacPlan planner is used with a path allocation approach where it cannot allocate individual plans to different agents. This approach requires that each occupant at a node receives the same plan (whichever of the several plans produced for that node would have gone to the majority of occupants):

- Group Plan Allocation (all agents at node receive identical plan)

Hazard location is fixed in the centre of the B2 building from Chapter 4 (with fully coupled graph), with 4 exits at the perimeters of the building. In the experiments, the hazard spreads from the centre to the other rooms (Figure 62) at a slow rate (0.5m/s).

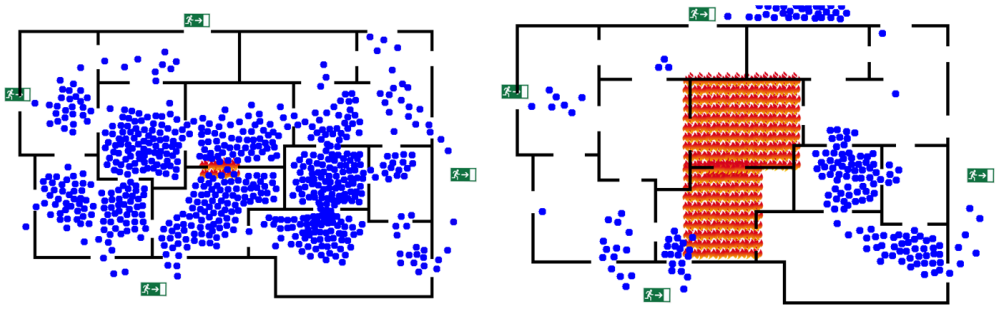


Figure 62: Evacuation progress at 179 ticks and 629 ticks (using Proximity Allocation without delay, "prox")

For the comparison experiments using Shortest Path plans, each agent receives individual plans for exiting the building; but as the shortest path from any node is identical for all occupants of that node, this is analogous to Group Plan Allocation without Delay but without accounting for capacity and congestion.

5.9 EvacPlan Experiment Results

For each population size (100–900 in increments of 100) an emergency evacuation of the B2 building is performed. The agent starting locations are duplicated for each Plan Allocation strategy, as well as for the Shortest Path evacuation scenario to ensure fair comparison. In each case, the time taken for the 95% of the uninjured population to evacuate ("Time to 95", as per [86]), and the number of injured occupants is noted.

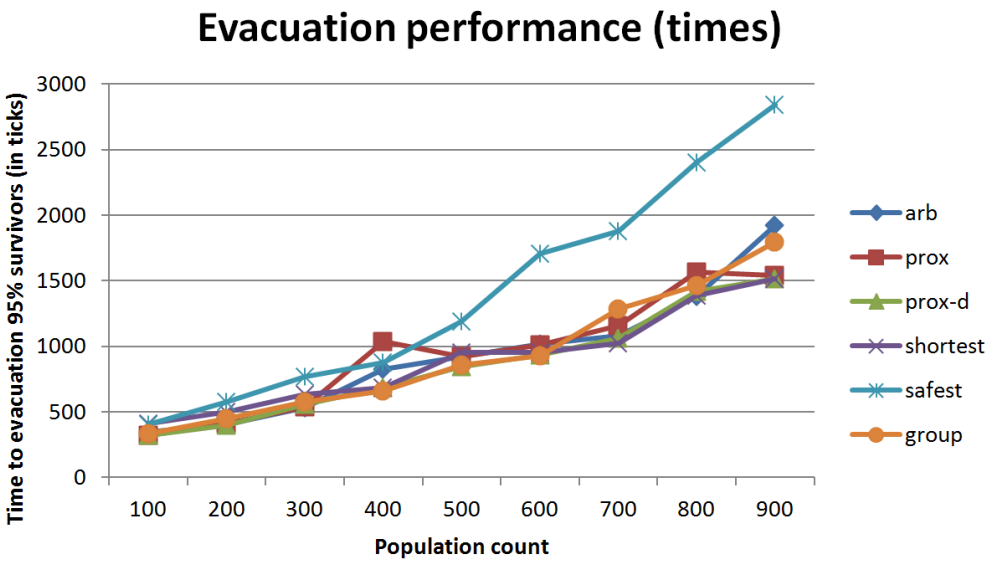


Figure 63: Experiment results (evacuation time)

The performance (Figure 63) of Safest Path is the worst among the evacuation planning options when it comes to evacuation time, a result of the high degree of caution employed by this method, which takes long paths that maintain distance from locations near the hazard. Proximity Allocation without Delays (*prox*) and Arbitrary (*arb*) both feature a spike in traversal time at 400 occupants, a result of the evacuation solution provided by EvacPlan resulting in a significant contraflow problem at one of the doorways in the eastern part of the building (Figure 64). Proximity Allocation with delays (*prox-d*) avoids this contraflow event by placing delays on the additional agents that would have resulted in the temporary deadlock, allowing the first flow to pass through before the second flow has arrived.

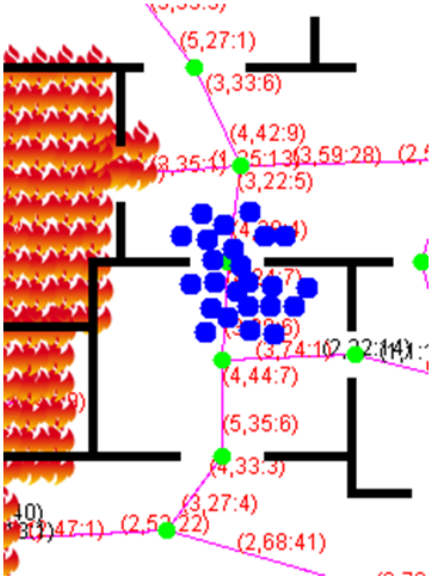


Figure 64: Temporary deadlock at doorway for 400 agents using prox

In the averaged results (average for 100–900 occupant evacuations, Table 9) we can

Table 9: Average Evacuation Time for evacuation experiments

Strategy	Time (ticks)
arb	936
prox	945
prox-d	860
shortest	897
safest	1405
group	928

see the trend of the Proximity Allocation with Delay (*prox-d*) performing the best of the dynamic flow-based evacuation strategies. Proximity Allocation without Delay (*prox*) performs more poorly on average than Arbitrary (*arb*) due to the spike at 400 occupants; but otherwise achieves better results.

As expected, the highly conservative safest path strategy gives the poorest evacuation times, and while the shortest path strategy performs well for low populations, congestions begins to impede its performance as it does not choose routes with consideration of limited capacity, leading to queuing which causes significant delays. The group path allocation approach gives quite low average evacuation times despite the reduced control over occupant path allocation, but as we will see in Section 5.9.1, this is at the expense of occupant safety.

5.9.1 Injury count

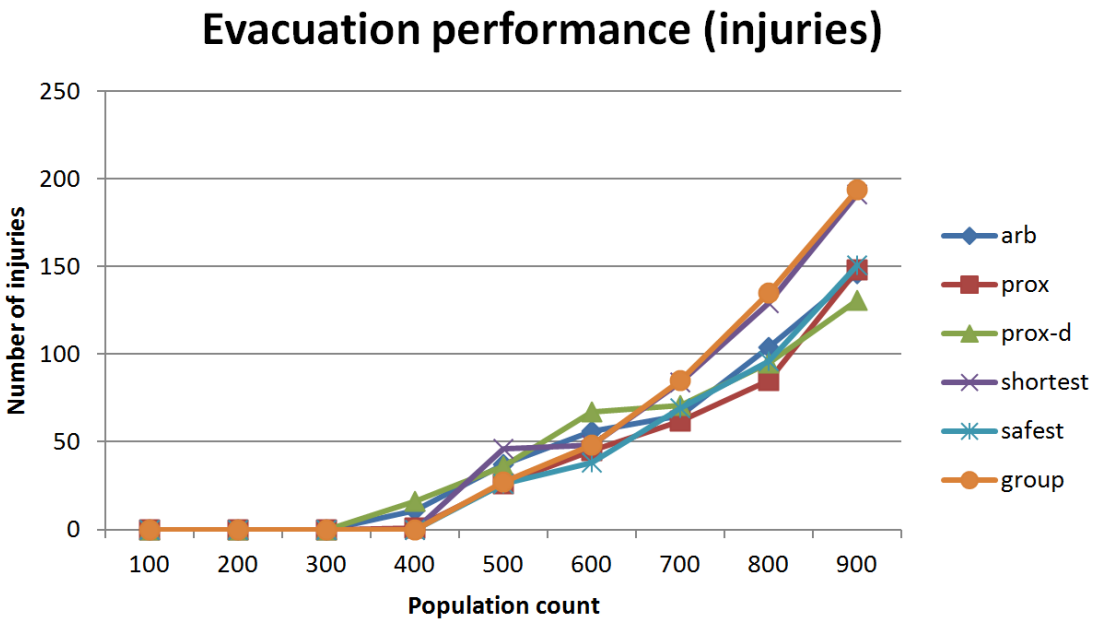


Figure 65: Experiment results (injury counts)

In each of these experiments, no injuries were encountered at population counts

Table 10: Average number of injured agents at time of 95% evacuation

Strategy	Average number of Injuries
arb	47
prox	41
prox-d	46
shortest	55
safest	42
group	54

below 400 (Figure 65); with Shortest, Safest and Group having no injuries below 500 agents as these plans tend to avoid edges that are about to become hazardous soon: Safest avoids these edges generally, and Group puts all occupants on a start node onto the same route which is often also the one that avoids edges that will become hazardous. Shortest performs well on injuries for these lower population counts simply because it will clear the building the most quickly when congestion is less of a factor, but soon results in greater casualties as the population size increases and queuing in areas that become hazardous leads to injuries.

Unsurprisingly, the congestion and hazard-unaware planning strategy of Shortest path performs the poorest on injury counts at higher populations. The intelligent flow-based plans had slightly increasing injury counts as the population size increases, though these occasionally outperformed the Safest strategy due to the quicker clearance of areas possible when distributing the building population more evenly based on path capacities. Proximity Allocation without delay (*prox*) outperforms Arbitrary allocation (*arb*) on injury counts, a result of the more orderly initial clearance of starting positions provided by this strategy. The additional delays used by Proximity Allocation with delay (*prox-d*) performed less effectively than without (*prox*), or arbitrary (*arb*), suggesting that requesting occupants to wait before starting their path is not an effective approach achieving smooth evacuation. Due to the method that evacuation times are calculated (time taken for 95% of the uninjured population to clear the building), the lower evacuation times for *prox-d* can in part be attributed to the lower uninjured population remaining in these cases, relative to *prox* and *arb*.

Group path allocation (where every occupant at a node receives the same path) performs similarly to shortest path; while these approaches give smooth flows of occupants (as every occupant in a group will be heading in roughly the same direction), the group allocation approach discards much of the capacity-sensitive power of EvacPlan and causes an excessively large population of the agents to travel in the areas near to hazard.

Averaging results between the runs (100–900) for each strategy (Table 10), we can

conclude that Proximity Allocation without Delay (*prox*) achieves the best results on average for injuries. While *safest* performed well at low populations, at 700 and above it performed poorly as the congestion effects limited the rate at which this approach could successfully clear the population. *arb* slightly underperforms relatieve to *prox*, and despite the spike in evacuation time encountered by *prox*at 400 occupants, it still outperformed *arb* on injury count at this population size. *prox-d*begins to result in slightly elevated injury rates to the strategies that ignored delays, due to the additional time taken to clear the initial positions of the occupants, causing some early injuries that are not as prevalent with *arb* and *prox*.

5.9.2 Evacuation performance over time

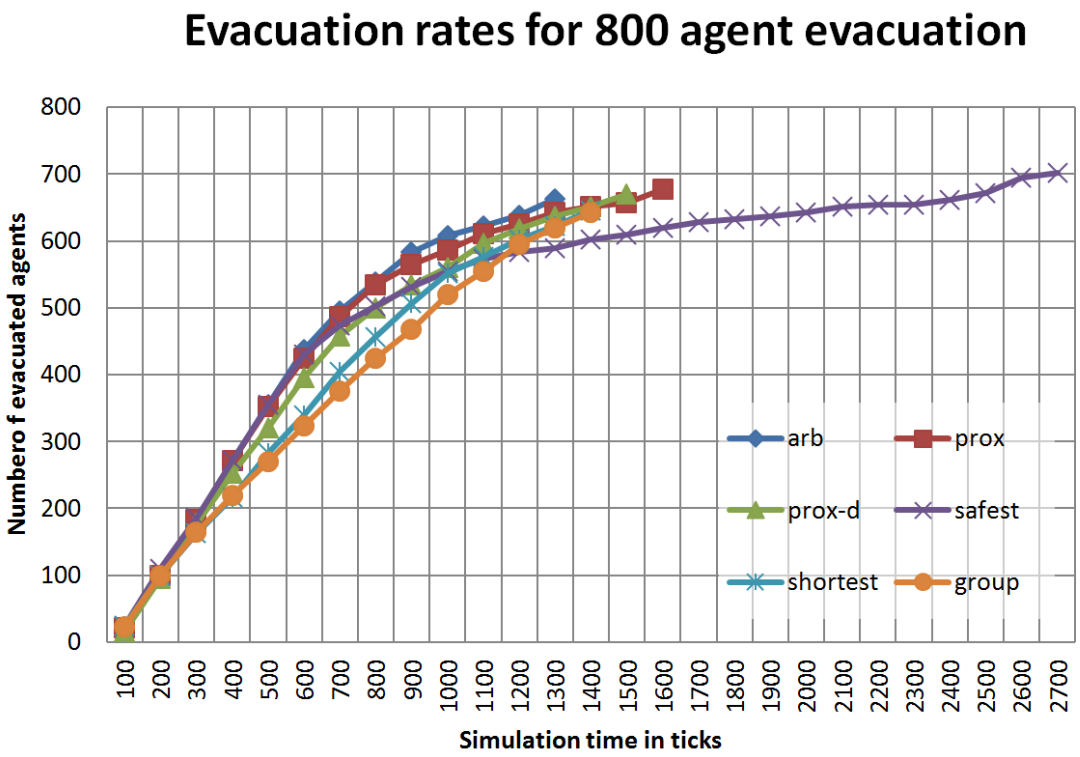


Figure 66: Evacuation rates for 800-occupant problem

An alternative perspective on the evacuation rate for the different planning and allocation strategies is shown in Figure 66. This figure shows the number of occupants successfully evacuated for 100 ticks of the simulation for the 800-agent evacuation task. The *safest* approach performs well early on but after 700 ticks or so it begins to slow down considerably relative to other strategies. The *shortest* approach, which does not account for congestion or hazard location, is consistently slower than the congestion-aware plans throughout the evacuations. Rate of evacuation for *prox-d* (nearest first with delay) is generally lower than the other dynamic plan allocation strategies, due to the delay in departure of occupants early on having a knock-on effect on the rate of travel through the building and out.

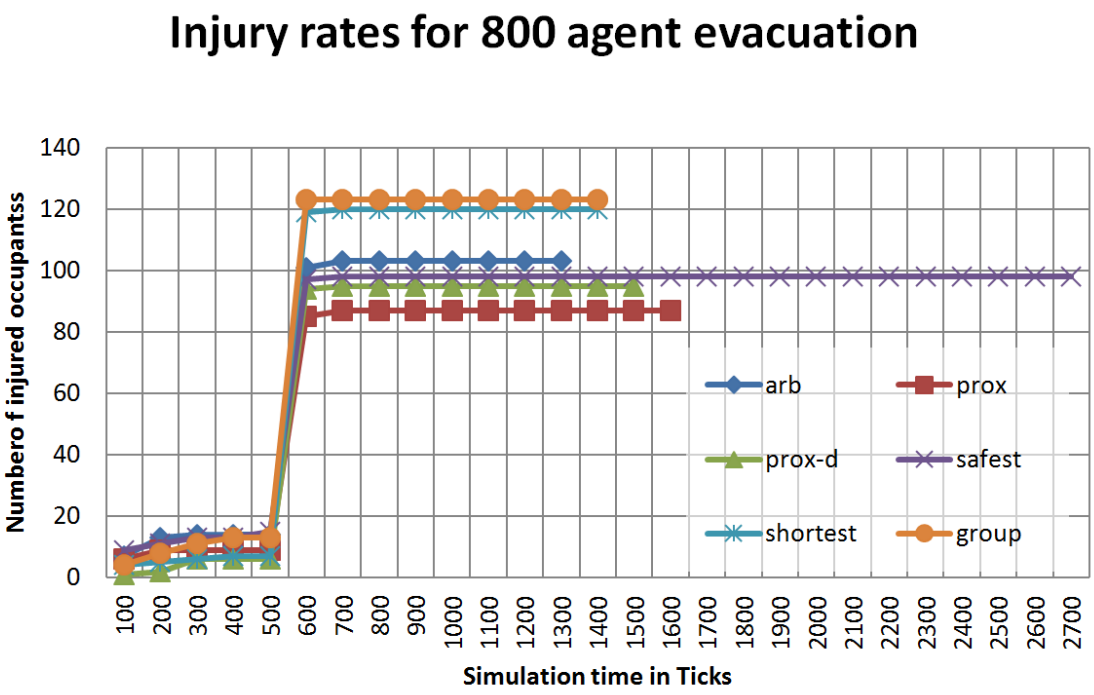


Figure 67: Rate of injury for evacuation of 800 agents

In Figure 67 we show the number of injuries at each timepoint, showing that for all strategies, there is a large spike in injury count at 500-600 ticks, a result of the hazard spreading into the area near to exit at the bottom of the building, and after this point there are few additional injuries. Group path allocation achieves the highest values, with *shortest* performing only slightly worse. While the paths allocated to groups are generated with congestion and hazard location in mind, EvacPlan's algorithms expects the group to split in different directions; when the occupants disobey this and stick together performance is very poor.

Here we see proximity allocation without delay (*prox*) providing the best overall result in the 800-person problem; while the rate of evacuation is similar to arbitrary allocation (*arb*) at 600 ticks (Figure 66), *prox* and *prox-d* have fewer injuries prior to this point due to the smoother early departure of agents these can provide. *prox* and *prox-d* also manage to clear a greater number of agents through the time-sensitive area that the hazard spreads into, resulting in fewer occupants getting caught in the hazard. *prox-d* features slightly higher injuries than *prox* after this point due to the time lost through the delay resulting in a greater number of agents being still present at the area into which the hazard spreads later into the emergency.

5.10 Conclusions

By simulating the movement of pedestrians in a two-dimensional free space pedestrian simulator, we can determine the throughput and walking speeds of crowds

through particular spaces. By simulating the movement of crowds along spaces associated with edges on a network flow graph, we can discover appropriate flow parameters for this graph. Through experiments, we showed that determining flow for edges on an individual basis produces accurate throughput and traversal time estimates on multi-edge paths through the graph by comparing these estimates with simulated crowds moving along several edges in sequence. These coupled graphs correspond well with the microscopic simulation results (which in turn is based on the agent model validated using real-world pedestrian data in Chapter 3). We discovered that small error in capacity estimates over paths tend to balance out, but that the macroscopic flow graph alone did not provide sufficient detail to predict the small errors.

By coupling the macroscopic graph flow parameters to the simulated motion of microscopic agents along those flows, the macroscopic model can reflect some of the nuance of pedestrian space use that would not be possible using basic building safety guidelines for space capacities and traversal times. The flow graph attains some of the realism of the microscopic multi-agent simulation while maintaining its strengths in computational and analytical simplicity and scalability. We demonstrated the predictive power of coupled graphs, showing that the graph parameters can be used to predict the first-and-last arrival times for groups of agents traversing the building space in EvacSim. As the graph parameters reflect the results of EvacSim simulation, the graph can easily be analysed to make predictions about group movement in the microscopic simulation, allowing for analytical reasoning and prediction about microscopic agent simulation that would otherwise not be feasible, or inaccurate (e.g. traversal time based on straight line distances rather than flow graph parameters).

In Chapter 6 we will look at exploiting the predictive power of Coupled Flow Graphs to increase the scalability and performance of EvacSim, by using this kind of Graph-Based prediction to subdivide the simulation world based on the likelihood of occupants being in proximity to each other in the near future. Exploiting the coupled graphs in this way allows for scalable simulation of multiple future states in short time-frames, enabling the development of new evacuation sensor reporting approaches based on predictive scenario simulation.

In this chapter we also investigated some path configurations that demonstrate some limitations of Network Flow Graphs in general; in these cases the traversal time estimates overestimated the achievable times as the graph cannot model the interaction of separate flows without significant modification. In general, the EvacPlan implementation of Hadzic's planner [33] performed quite well in evacuation problems. While evacuation plans that maximize safety above all else tended to produce low injury counts, they also led to high evacuation times which increases the risk of exposure for the occupants. Similarly, the Shortest path planning approach provided

low evacuation times (to a point) but at the expense of occupant safety, and due to the inability of Shortest Path to account for the limited capacity of building space, this approach becomes progressively worse as the building occupancy increased.

The three dynamic path allocation strategies (Arbitrary, Proximity Allocation and Proximity Allocation with Delay) each produced encouraging results for evacuation time and injury counts. As might be expected, by allocating routes to agents based on their proximity to the next step first, a group of agents that needs to be split in half can easily cleave down the middle and head in opposite directions without causing a contraflow situation at the beginning of the evacuation. Conversely, allocating paths arbitrarily to the occupants at a node gives a poorer result as this initial first step is chaotic as occupants shuffle through each-other to reach their respective next step.

A common issue in various instances when using the dynamic evacuation planner is its difficulty dealing with contraflow situations. In Section 5.6.1 we found that use of the building that causes contraflow (two flows in opposition) and junction crossroad (flows crossing a common central node simultaneously) are not modelled well in Network Flow Graphs and on many occasions this resulted in a long delay in parts of the building where these would occur, increasing the final evacuation time and occasionally resulting in casualties if the hazard has spread into the congested area. Despite the inaccuracies of macroscopic graph planning relative to the occupant movement, the congestion-aware planning consistently outperform alternatives when congestion is a factor in the evacuation.

The Proximity Allocation with Delays (*prox-d*) allocation approach performs the poorest of the three dynamic path allocation strategies on average injury rates over the range of population sizes, but in the high-population problems it was the best of the three "smart" plan allocation approaches with respect to this metric. Enforcing the delay on agents leads to a smoother use of the building space and is less likely to result in excessive queuing at bottlenecks as the overall flow in the building is kept smooth. When considering the additional "good behaviour" required of occupants to not only divide into groups as instructed, but also to wait when told to, these results that paths with delays may not be an appropriate solution. Furthermore, in the case of a sparsely populated building, it is preferable to discard the delays.

Path allocation using Proximity Allocation without delays (*prox*) produced good results overall, but, as with *prox-d*, requires a very fine accuracy of localization in order to achieve these results; it requires the localization of occupants and the direct communication to these occupants in order to operate. While this is easy to implement in simulation, real-world constraints limit the possibility of this kind of approach to path allocation. When considering very limited communication capabilities or the natural group-formation behaviour of occupants (using the "group" path allocation

strategy) the EvacPlan planner achieved poor results; though it should be noted that the plans EvacPlan generates for this purpose assume that it is possible to split the group up and the poor performance is a result of the significant deviation between this expectation and the simulation reality. When considering evacuation problems in this context (where there is a lack of fine control over occupant movement or localization, or when considering human nature), a different approach to dynamic evacuation planning is needed to consider these limitations. While EvacPlan and similar dynamic evacuation planners (Section 2.4.3) are designed to rapidly re-plan to respond to the evolving state of the building evacuation, replanning is of little use if the newly generated plans are subject to the same limitations as the original evacuation plans.

Considering the limitations of tracking and communication, planning strategies with these limitations in mind should be developed, for instance:

- Plans that can be implemented through building signposting rather than direct communication
- Plans that guarantee that everyone in a particular starting position is given the same plan
- Solutions that do not rely on initial delays as a flow control mechanism, as occupants are unlikely to obey delays

6 Exploiting Macroscopic Models for Microscopic Simulation and Sensing of Multiple Future States

6.1 Introduction

In Chapter 5 we investigated how macroscopic graph models could be improved by coupling with detailed microscopic simulation to allow for macroscopic-level reasoning to match closely with microscopic results. These results showed how macroscopic models are improved through integration with more detailed microscopic models, as demonstrated in traversal time predictions and the performance of flow-based dynamic evacuation planners. In this chapter, we reverse the relationship, exploiting these coupled Macroscopic Graphs to improve the microscopic multi-agent simulation.

In Section 5.5 we showed how, for individual groups, the coupled flow graphs were capable of accurate traversal time estimations for travel along paths, providing first- and last-arrival times that closely matched those achieved in microscopic simulation. In Section 6.2 we exploit these first- and last-arrival time estimates to make predictions, on the macroscopic model, about when and where different groups of agents might meet each other in the building within the near future. As isolated groups are predictable, they need not be simulated within the same simulation instance as other groups, and by dividing simulation space based on predicted interaction, we can reduce the complexity of simulation by reducing the number of agents present in any given sub-simulation.

As we showed in Section 5.5, isolated groups are predictable but when different groups meet each other in the building space, the impact on traversal times is volatile. Furthermore when accounting for the possibility of information exchange and changing paths, the interaction of two groups (forming a new group) could lead to a variety of different outcomes depending on which path the merged group follows. By sub-dividing simulation based on predicted interaction of groups, we can reduce simulation complexity while allowing for a reasonable number of branching future states, a powerful capability for emergency simulation in short time-frames.

Finally in Section 6.3 we describe simulation-derived Sensor Pattern Matching, an approach that takes the simulated multiple potential future states of 6.2 and generates sensor "fingerprints" associated with each simulated future branch. As the different future states might differ significantly from that expected by dynamic evacuation planners (and require re-plans), by producing the re-planned evacuation routes ahead of time for each future state the new plans can be ready immediately enacted when observations match precomputed states. Using this approach, we can exploit the predictive capability of simulation of multiple future states to respond to the

evolving state of the emergency with pre-prepared contingency plans. This approach allows for quick response and ahead-of-time simulation within tight time-frames, and also allows for pattern matching and dynamic guidance on-scene in case of loss of connectivity with central computation facilities.

6.2 Problem Decomposition

Combining the evacuation plan allocation of 5.7 (allocating plans to individuals in the building) with the traversal time estimates of 5.5, we have the parameters for prediction of future locationss for agents as they follow their routes, provided that they do not interact with groups on different paths later in the building (which can lead to inaccurate prediction results, as we found in 5.6.1). By combining the information about the locations visited on routes with the number of occupants allocated the same route, we can group occupants according to their allocated paths and use these groups, with the macroscopic flow graph, to determine the First- and Last-arrival times at each node on the route.

In Section 5.6.1 we found that for single groups these predictions were accurate, but when multiple groups meet each other in the building (such as at T-junctions or in contra-flow situations) the traversal estimates are unreliable. Furthermore, when different groups meet in the building they are likely to form a larger combined group rather than maintaining their original routes. When two groups combine, the traversal prediction procedure needs to account for the multiple possible routes that the combined group can continue on (if the remaining routes are different) as well as the larger size of the new, combined group. For large numbers of groups and routes, this results in explosive growth of possible future states where groups meet and combine and follow different routes than those that they started on (Figure 68).

In Appendix A.4 we show how the performance of EvacSim scales based on the quantity of agents simulation, and also their density. When groups of agents in one area of a building do not interact with agents in another (such as when their paths don't meet, Figure 69, or one group passes through a space some time before another), they can be safely simulated seperately. In A.4.4 we showed performance improvement achieved by geospatial locality indexing, which maintains an index of agent locations and uses this to avoid inter-agent calculations where the agents are not adjacent in space (spatially orthogonal). By using the macroscopic graph to predict when groups are present and at which nodes, we can determine which occupant groups are orthogonal to each other in time *and* space and simulate them seperately (and hence reduce the amount of inter-agent computation and limit the number of future states considered).

Decomposing the problem in this manner and computing it in parallel would enable

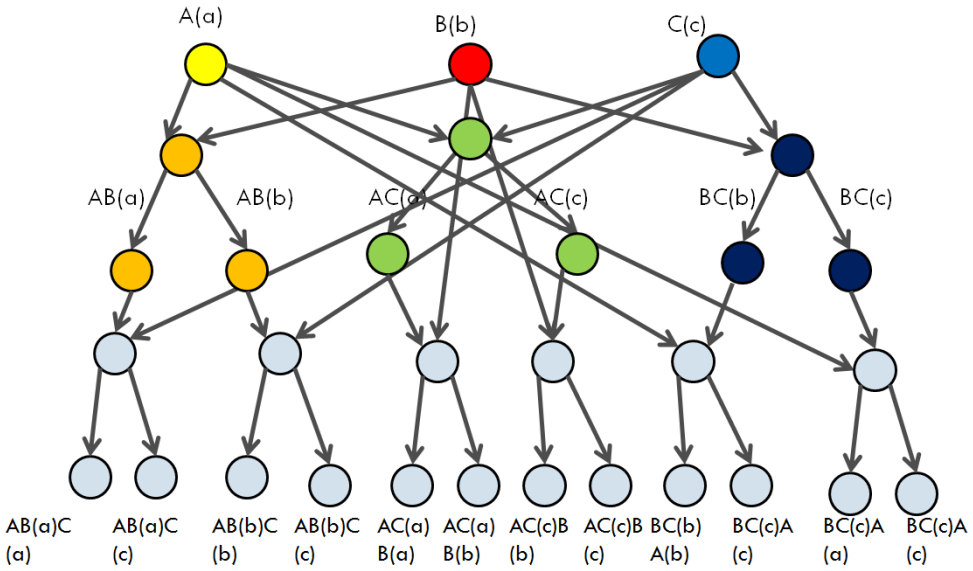


Figure 68: Expansion of potential future states for 3 groups meeting in the building

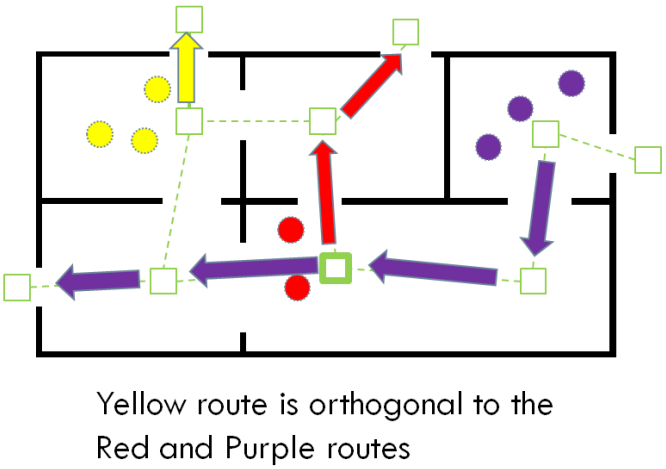


Figure 69: Orthogonal Paths

many more possible futures to be simulated in a time-frame that allows for predictive feedback to fire-safety personnel during the emergency. Further, by handling the uncertainty of evacuation scenarios using an agile simulation strategy, we responding to new evacuation data and quickly decomposing the problem space to drive a new cycle of simulation.

6.2.1 Overview and Assumptions

Our approach is to perform a series of filtration operations to produce the parameters for a set of several simulations. As there are limitations on time and computational resources, we make some simplifying assumptions as to occupant behaviour and analyse the building structure to identify where occupants are likely to meet each other in the building. While occupants during emergencies are inherently unpredictable, we believe it is pragmatic to disregard the possibility of particularly unusual be-

haviour (such as occupants heading away from exits or splitting up groups) until we receive information that this unusual behaviour is actually occurring in the building via sensor data or eyewitness reports.

By analysing the building structure (in the form of a network flow graph), we can determine which occupants in the building are orthogonal in time and space before we go to the trouble of simulating them together at once. Spatially and temporally orthogonal groups of occupants can be simulated in separate simulation instances as their members are not predicted to interact with each other. For instance, occupants on opposite ends of a building are not likely to meet each other when evacuating, and can be safely simulated separately without concern for modelling interactions between the groups.

To determine which occupants interact with each other, we first assume that occupants that begin the evacuation together on the same route form a group. We assume this group will stick together and may follow one of several routes out of the building, either those allocated by an evacuation planner, or chosen by the group's own decision making procedures. We also assume that when separate groups meet each other during the evacuation, they merge together and continue using one of the constituent groups' routes. By grouping occupants in this manner we reduce the problem complexity significantly, as we focus on permutations of group decisions rather than of individual occupant decisions.

6.2.2 Route Allocation

Occupant groups have a number of options as to how they may proceed out of the building; in fact if we allow looping there are an infinite number of routes to exits. To reduce the problem space, we make some assumptions about likely occupant behaviour. We assume there are a limited number of routes the occupants are likely to take; in this work we make use a hazard- and congestion-aware evacuation planner (EvacPlan, as used in 5.7) to produce routes which are communicated to the occupants using the "Proximity Allocation without Delay" (*prox*) path allocation of Section 5.8.6 . This evacuation planner accounts for the location of hazards and produces evacuation routes that minimize evacuation time and exposure to the hazard. Alternative route generation schemes could be used, such as group-level agent planning or exploiting day-to-day sensor data to discover common exit routes, approaches which we consider for future work.

6.2.3 Complexity and orthogonality

The number of potential futures simulated is contingent on the number of possible routes the occupant groups could take, and the amount of interaction between groups. As we assume groups that meet in the building could proceed according to either constituent group's route, there is the potential for many branching future states (Figure 68). In this figure just three initial groups (A on route "a", B on route "b" and C on route "c") could potentially meet at many different times and decide to take many different routes from those meeting points after merging together (AB(a) meants C(c) and produces AB(a)C(c) using the "a" route or the "c" route").

In this research we perform initial analysis of the Network Flow Graph and Routes in order to filter out combinations which are unlikely to occur, and only simulate the remaining combinations. By exploiting the Network Flow Graph to determine the times at which crowds arrive at nodes, as well as the duration they spend at the node, we can identify when and where crowds might meet each other. Crowds that do not meet at any time in the graph are considered to be spatially and temporally orthogonal to each other and do not need to be simulated together (Figure 70). This characteristic is crucial to reducing the problem space, as we do not need to simulate the product of the permutations of multiple orthogonal crowds.

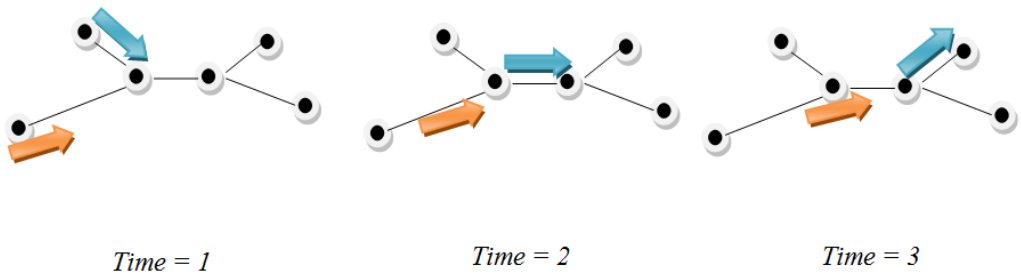


Figure 70: Two crowds occupying the same space but at different times are orthogonal to one another

In the case that multiple groups meet each other in the building, we need to account for the branching possibilities that proceed from the interaction. Where crowds meet, they interact, share route information and lead to greater queues and delays on the groups. In our approach we merge the groups together and investigate the branching futures where they proceed along any one of the original groups routes. For instance 3 groups of 10 meeting leads to investigation of 1 group of 30, and the 3 possible routes it could take (as drawn from the original 3 groups).

6.2.4 Discovering Orthogonal Future States

Crowd-Route generation

Each node in the Network Flow Graph has an initial occupancy value, reflecting the presence of occupants as reported by sensors. This occupancy information initializes a "Crowd" object for each occupied node in the graph. EvacPlan provides sets of likely routes which could be followed by occupants at nodes, each combination of a route with a crowd generates a CrowdRoute object. For example, twelve occupants at a node have three routes assigned to their members. As we assume that the Crowd does not split up, this results in the generation of three CrowdRoute objects of size twelve, accounting for the three routes that the group of occupants could take in exiting the building.

Time expansion of Network Flow Graph (TimeGraph)

In this scheme, we filter out impossible future states of the evacuation within a given time period. To achieve this we divide the time period into individual timepoints (at 15-tick intervals) and each Node has a list of occupancy records, which will be used to represent occupancy of the Node at each timepoint. This allows us to recognise that two groups passing through the same node at different times are orthogonal in time and hence do not collide. From now on, we refer to this time-expanded network flow graph as a "TimeGraph".

First- and Last-Arrival calculation

We determine that two CrowdRoutes interact if they ever occupy the same space at the same time. To determine if this is the case, we work the CrowdRoute through its route on the graph, determining the earliest and latest arrival time at each node during its journey and noting the presence of the CrowdRoute at each Node for any timepoints between these two values. The earliest arrival of a member of the group at each node is given by the traversal time over the connecting edge. The latest arrival time is dictated by the size of the Crowd and the capacity of the edge, (as discussed in Section 5.5). As such, a narrow corridor may take longer for a large crowd to traverse than a wide open space, due to queuing. This queuing leads to members of the CrowdRoute arriving at the next node over the course of an extended period and they are more likely to collide with other CrowdRoutes as a result. Note that this computation is carried out on the TimeGraph before we execute any expensive multi-agent simulation.

Algorithm 8: First and Last Arrival calculation

```
Data: (TimeGraph) TG, (Set) CrowdRoutes
(int) maxdur = 30;
(int) T = 0;
foreach CrowdRoute CR ∈ CrowdRoutes do
    (Node) n = route_0;
    (Node) m = route_1;
    (int) i = 1;
    while i ≠ route.size() do
        (Edge) e = TG.getConnectingEdge(n,m);
        (int) firstArrival += (E.minTraversal + E.maxTraversal)/2;
        (int) lastArrival += firstArrival + CR.size/E.capacity;
        for (int) p = T+firstArrival; p < T+lastArrival; p++ do
            n.addAtTime(p,CR);
            if p < maxdur then
                m.addAtTime(firstArrival+p,CR);
            end
            T = T+firstArrival;
        end
        m = n;
        i++;
    end
end
```

CrowdRoute collision discovery and merging

Having traced each CrowdRoute through the graph (along with any new CrowdRoute objects generated as a result of merges), we can then examine each timepoint on each Node to determine if and where CrowdRoutes simultaneously occupy a node. In cases where CrowdRoutes interact, we merge the CrowdRoutes together, and repeat the "First-and-last-Arrval" calculations using the new merged CrowdRoute (unless the CrowdRoutes involved contain the same Crowds as each other, as a Crowd cannot collide with itself).

In the case of a merge, all the colliding CrowdRoute Crowds are combined together into a combined Crowd object. This Crowd has a size equal to the sum of the constituent crowd objects. Each original CrowdRoute's Route is used in the creation of a new set of CrowdRoute objects which are inserted into the TimeGraph nodes at the timepoint of the earliest first-arrival among the colliding CrowdRoutes at that node. The merged CrowdRoute contains a record of the constituent Crowd objects and the routes they were following before the merge occured.

Example:

- CrowdRoute **A-a**, **B-b** and **C-c** collide at a node.

This produces three new CrowdRoute objects (one for each route they proceed on):

- CrowdRoute **A-a, B-b, C-c(a)**

- CrowdRoute **A-a, B-b, C-c(b)**
- CrowdRoute **A-a, B-b, C-c(c)**

The new CrowdRoutes are inserted into the graph at the node the collision occurred, at the earliest timepoint any of the constituent CrowdRoutes was present at that node and the first-last arrival operations (Algorithm 8) are performed for these new merged CrowdRoutes. This may in turn lead to the discovery of more collisions and generate further merged groups; this continues until no new collisions are discovered (Algorithm 9). The maxdur value provides a natural termination point for this cycle, indicating how far into the future we intend to simulate.

Algorithm 9: Collision Detection and Merge Operation

```
Data: (TimeGraph) TG
foreach Node n ∈ TG do
    foreach Timepoint T in N do
        if |T.contents| > 1 then
            if !(exists(CrowdRoute(T.contents))) then
                (int) arrival = earliestArrival(N,T.contents);
                (Set) Routes = getAllRoutes(T.contents);
                Routes.removeDuplicates();
                Routes.startFrom(N);
                foreach Route R ∈ Routes do
                    (CrowdRoute) merged = new CrowdRoute(T.contents);
                    merged.setRoute(R); N.addAtTime(arrival,merged);
                end
            end
        end
    end
end
```

6.2.5 Instance generation

The Collection of all CrowdRoute objects generated during the collision detection and merging phases represents the possible ways occupants in the building could meet and proceed together. Each CrowdRoute generated as a result of a merge features the constituent Crowds as well as the routes that they took in arriving at the node at which the merge occurred. CrowdRoutes generated from the merging of other merged CrowdRoutes also hold the information describing which route they followed after each previous merge. For each CrowdRoute, we can create a seperate EvacSim instance.

Each instance contains occupant agents representing the members of the Crowd objects present in the CrowdRoute. Each occupant agent is given a sequence of routes to follow; initially following their original route (e.g. occupants from CrowdB

Table 11: CrowdRoute merging example

CrowdRouteA-a CrowdRouteB-b CrowdRouteC-c	the original three CrowdRoutes
CrowdRouteA-a B-b(a) CrowdRouteA-a,B-b(b)	A meets B
CrowdRoute[A-a,B-b(b)],C-c(b)	A+B on B's route meets C, uses b
CrowdRoute[A-a,B-b(b)],C-c(b)	A+B on B's route meets C, uses b
CrowdRoute[A-a,B-b(b)],C-c(c)	A+B on B's route meets C, uses c

follow route b) until they meet occupants from other Crowds in the simulation space. When they meet, they proceed according to the record in the compound CrowdRoute object.

As an example, following the collision and merge operations, we may be left with 7 CrowdRoutes (Table 11, and illustrated in Figure 71):

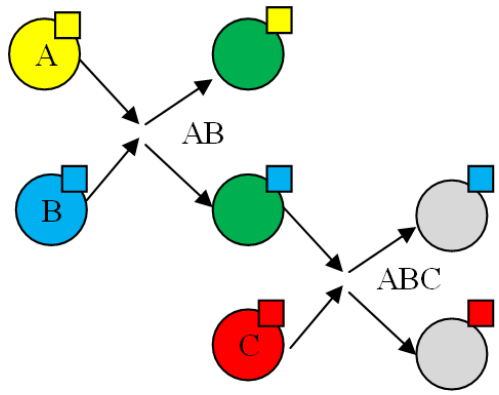


Figure 71: Example of CrowdRoute merging. Coloured squares indicate the route the CrowdRoutes are using

This leads to the generation of 7 seperate EvacSim instances, one for each original CrowdRoute object and one for each merge operation that occurs. This accounts for groups merging, and allows for the possibility that two groups might fail to meet each other.

6.2.6 Decomposition Experiments

These experiments were performed using a model based on the 3rd floor of the Kane Science Building at University College Cork with initial populations of 85 and 340 occupants (giving a sparse dispersal of occupants in the space, as well as a denser dispersal with 4 times the density) spread among various office and lecture spaces, constituting 19 initial occupant groups (Figure 72). This population features a

mix of large and small groups, and the floor layout produces a variety of routes to exits and varying flow capacities due to the variety of corridor widths and room sizes (Figure 73). This floor features six exits and in this experiment we produce orthogonal simulation sets based on a 40-second prediction period. We repeated this experiments adjusting the population density and varying the location of the hazard to produce a variety of different initial states and outcomes.

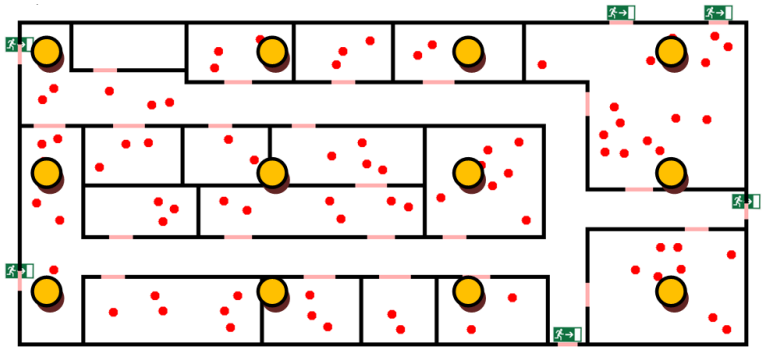


Figure 72: Experiment building with occupants (small circles) and hazard start locations (large, outlined circles)

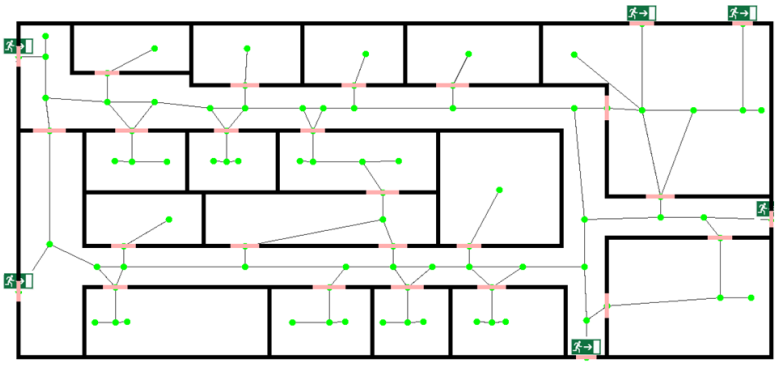


Figure 73: Experiment building with graph

The initial fire was placed in one of 12 starting spots drawn from a 4x3 grid (Figure 73). In each case, we noted the running time for the decomposition procedure, the number of orthogonal simulation instances produced, and the largest and average size of each simulation instance in terms of constituent groups. These experiments were repeated for an 80-second look-ahead period, to determine what impact increasing the duration of time investigated had on the number of simulation sets and on runtime of the problem decomposition routines.

6.2.7 Decomposition Results

We recorded the decomposition running time and the number of simulation instances suggested in each case (Table 12). For comparison, we reproduce the EvacSim run-

Table 12: Problem decomposition experiment results

	Runtime(seconds)	Number of instances	Occupants per instance
85 occupants, 40 second look-ahead	Average: 0.1420 Std. Dev. : 0.0984	Average: 25.8 Std. Dev.: 1.5	Average: 5.7 Std. Dev. : 0.4
85 occupants 80 second look-ahead	Average: 0.1724 Std.Dev. 0.117	Average: 29.8 Std. Dev. 7.2	Average: 6.8 Std. Dev.2.3
350 occupants 40 second look-ahead	Average: 0.3531 Std.Dev: 0.1875	Average: 40.2 Std. Dev.: 2.3	Average: 32.4 Std. Dev.: 1.5
350 occupants 80 second look-ahead	Average: 0.9741 Std.Dev: 1.0832	Average: 64.9 Std.Dev: 8.2	Average: 51.2 Std.Dev: 8.3

ning time results from A.4.5, to gauge the impact of running times of the simulation instances (Figure 74)

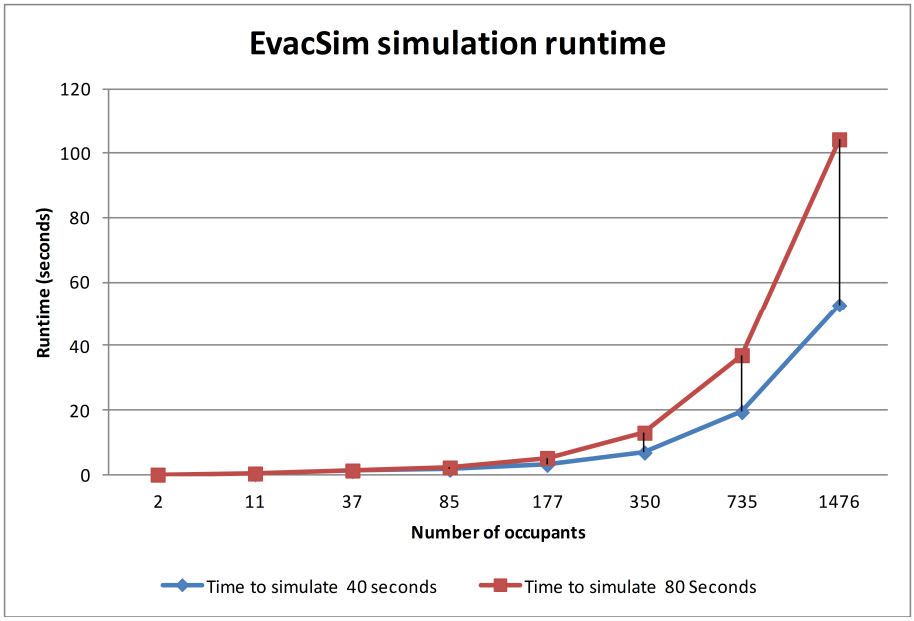


Figure 74: EvacSim running time for 2-1476 agents

For a population of 85, with both durations, we found that EvacSim simulated the entire population in a fraction of wall-clock time, 1.5 seconds for a 40-second simulation look-ahead period and less than 3 seconds for an 80-second period. Similarly we found that EvacSim was capable of producing simulation results at a fast rate for the population of 350, with runtimes of 7 seconds for simulating 40 seconds of evacuation, and 13 seconds for an 80-second simulation period.

We found that the decomposition method suggests average simulation instance counts well below 100, and for the lower crowd density and lookahead period the

number of instances was very low. For the most part these simulation instances feature a population count which is a fraction of the total building occupancy, suggesting running times between 0.4 and 1 seconds for each simulation instance. For long look-ahead periods, the tendency is for crowds to merge together more often as queues begin to form at exits. We found that for an 80-second look-ahead period, the average number of simulation instances is higher relative to 40-seconds, reflecting the tendency for large merged groups to form as CrowdRoutes converge on exits.

For some hazard positions, only a limited subset of the building exits appear in evacuation routes as the hazard blocks some escape routes. In these cases, there is a tendency to produce a large merged group as the majority of occupants head towards fewer remaining exits. In these cases, there is a greater runtime for the problem decomposition software relative to other hazard positions (3 seconds vs 0.2-0.4) as there is a greater amount of crowd merging operations to compute. A hazard spawning in the leftmost position in the second row cuts off a number of exits. In the experiment using 80-second look-ahead and 350 occupants, this led to 80 EvacSim instances and with an average of 62 occupants in each one (Table 12). This result suggests that a balanced decomposition of the problem might not always possible using this technique alone, as in some scenarios (e.g. limited exits or highly restricted flow) there is a low degree of orthogonality.

Hazard position has a significant impact on the balance of the simulation sets; a single high-population simulation instance is more computationally expensive to compute than a small set, and may constitute a simulation bottleneck as we wait for the large set to finish simulating before we can make use of its prediction results.

6.3 Simulation for Event Prediction and Sensing

The safety and speed of emergency evacuation of buildings can be improved with the exploitation of networked smart building surveillance (Appendix A.3.8) and evacuation control components and centralized evacuation planning algorithms (Section 2.4.3). Monitoring of smart buildings is often performed using wireless sensor nodes which are battery-powered and have limited computation capability. In the emergency evacuation scenario, wireless sensor networks (WSNs) monitor the occupancy of the building and location of hazards (fire, smoke, chemical spill) and this information is combined with knowledge of the building layout to generate evacuation plans.

The plans can be computed at a centralised Base Station (BS) using an optimal evacuation plan algorithm based on initial occupancy, knowledge of the building structure and knowledge of the behaviour of the hazard. The evacuation plan can

be enacted in the building in the form of a set of instrument actuation instructions, such as direction signs configured to direct occupants along the optimal route.

Timely response to events in a building during emergency is of key importance to ensure sensible evacuation planning; if egress of the building proceeds in an unexpected manner, the system needs to respond quickly and generate an updated evacuation plan to ensure the safe evacuation of the occupants. Network communication combined with computation of updated plans may lead to plans which are already out of date by the time they are received by nodes in the network. Our goal in this research is to reduce the communication burden on sensor nodes and improve response time to changes in evacuation progress by using a resource-rich Base Station (BS) node to predict future movement of building occupants.

By performing Multi-Agent Crowd Evacuation simulation at the BS or connected server we simulate several possible future progressions of the evacuation. The BS simulates how each of these candidate scenarios would be perceived by the sensors and computes the optimal evacuation strategy in each scenario, and then sends this information to the wireless sensor nodes. As the evacuation proceeds, the sensor nodes use this information to identify the simulated scenario that best matches their actual readings. If the sensors determine that one of the candidate scenarios matches closely to their readings, they can then inform the nearby direction signs as to the proper evacuation plan to enact without requiring a time-consuming multi-hop report to the Base Station.

6.4 Candidate Event Simulation

6.4.1 Assumptions

We assume that a structural model of the building is available and that the sensing coverage region and wakeup schedule of sensor nodes is known by the Base Station. We assume, before the emergency is reported, that the Base Station has built up a perspective of the building occupation density in each room, based on reports from the sensor nodes. We assume that an emergency alert is generated by the network (for instance, a smoke alarm triggers as a result of fire), and that this alert informs the Base Station as to the location and likely spread of the hazard. We assume that WSN nodes periodically report to the Base Station on the estimated occupancy of their sensing regions. We assume that the WSN includes actuation instrumentation (such as signposting or direct communication, cf. Sections 2.4.3 and A.3.8) which can be instructed with via wireless communication to direct occupants along particular routes. We assume that the nodes have timekeeping capabilities and that the Base Station and wireless nodes are loosely synchronised. We do not

assume that there is full sensor coverage of the building.

Progress of the evacuation may evolve in a number of ways, for instance occupants might all choose to head towards a particular exit, or a group of individuals might ignore the signpost guidance and take an unexpected route to safety. Occupant movement is limited by physical constraints such as inability to pass through obstacles, and is motivated by typical human behaviours such as forming groups that travel together, and avoiding danger.

In our approach (Predictive Simulation Reporting) the BS generates multiple possible futures ("scenarios"), covering short periods of time (e.g. 40-80 seconds) and produces a simulated set of sensor readings for each scenario. Each of these scenarios represents different combinations of routes occupants could take to safety, and variations in occupant response to the signpost actuation. The simulated readings are sent to the sensor nodes in the WSN ahead of time and the surveillance sensors are required only to compare their readings with the simulated readings to determine the best matching candidate scenario. The optimal evacuation plan for each scenario in the form of actuation instructions to actuation nodes in the network (e.g. via multi-hop wireless communication)

Our aim is for the sensor nodes to detect if the evacuation is progressing according to one of these candidate scenarios and inform the actuation nodes as to which evacuation plan they should implement. We also wish to reduce the communication burden for reporting evacuation progress to the Base Station by requiring the sensor nodes to simply report the degree to which each candidate scenario matches what they actually observe, rather than detailed periodic sensor readings. We believe that this approach will improve the response time in the network to changes in evacuation progress without significantly increasing computation or communication performed by the wireless sensor nodes.

The candidate scenarios are simulated using a multi-agent evacuation simulation (EvacSim) which simulates sensors and building occupants, generates the scenarios and produces these simulated sensor readings. These scenarios are generated based on initial occupancy data and different evacuation routes that groups of occupants in the building might take. Each simulation cycle produces a set of simulated sensor readings (based on observations of the simulated sensors) which is associated with that candidate scenario.

6.4.2 Candidate Event Confidence Calculation

Each sensor in the real network receives their set of predicted sensor traces and identifier of the associated candidate scenario for each trace. During the real evacuation, each sensor periodically compares their observations with the expected values for

each candidate event and determines their confidence in any particular event being a match for the real event (Algorithms 10 and 11). In the event of a good match, the sensor can report the ID of the best match to the Base Station. In the case of no good match, the sensor reports their historical readings to the Base Station which updates its view of the building based on the unexpected progression of the evacuation.

We assume that the occupant monitoring sensors are a simple binary motion detector with limited range. Such sensors detect the presence or absence of occupants within their range as a simple Boolean True/False value. A Sensor trace for these sensors is in the form of a time series of boolean values. These sensors are unable to determine the total number of occupants they can sense and cannot identify specific individuals; they merely detect presence of occupants within their field of view. This model was chosen as more sophisticated sensor types can be interpreted with this binary model (e.g. a complex video camera can categorise motion as a boolean variable).

The sensors compare their real readings with the predicted traces by determining the number of matching positive readings. Each candidate scenario is given a value based on the number of matches and the event with the greatest number of matches is chosen as the best matching scenario.

Algorithm 10: Trace Comparison Confidence (Single Reading)

```
Data: (boolean) a, (boolean) b
Result: boolean match
match = a ∧ b;
return match;
```

Algorithm 11: Trace Comparison Confidence (Total Confidence)

```
Data: (boolean)[] scenario, (boolean)[] readings
Result: (int) confidence
(int) confidence = 0;
for boolean r ∈ scenario do
    if Trace Comparison Confidence(r,readings[i]) then
        totalConf ++;
    end
end
return totalConf;
```

Confidence assessments are performed periodically on each sensor, taking all of the sensor readings since the last assessment and determining the total number of positive matches for each candidate scenario. Each such assessment can be reported to the Base Station as a sensor report, and if an individual sensor is particularly confident in a scenario (where that scenario has the highest confidence value and this value is greater than the next highest by a given threshold), they can instruct

nearby signpost nodes to implement the associated evacuation plan.

As confidence is based on the proportion of matching non-zero values, this approach is relatively robust to small changes in the start and finish of positive readings (e.g. occupants arriving late at a sensor). While the observations in a reading sequence might not perfectly synchronise with the scenario trace, there is still some overlap that generates positive reading matches and increases confidence in the scenario.

6.4.3 Pattern Matching Experiments

Two sets of experiments were performed to investigate the feasibility of the Predictive Simulation Reporting scheme. The first experiment investigates the time taken for individual sensors to determine a strong positive match to a scenario, and the number of correct and incorrect matches given by the sensors; the goal of this experiment is to determine the degree to which individual sensors can be relied upon to identify the matching scenario based on their own observations. In this experiment, the sensors identify a scenario match if it has at least 1.5 times more confidence in it than the next best scenario. The second experiment investigates the quality of matching when the Base Station receives the reports from sensor nodes and combines them together.

The experiments were performed in a simulated building (illustrated in Figure 75) with 7 rooms and an open common hallway space. A group of 20 individuals begins the experiment in the top-left room. 12 simulated motion-detecting sensors were placed in the building in a uniform grid. This deployment consisted of a mix of indoor and outdoor sensors in order to monitor the movement within the building and also observe occupants successfully exiting the building.

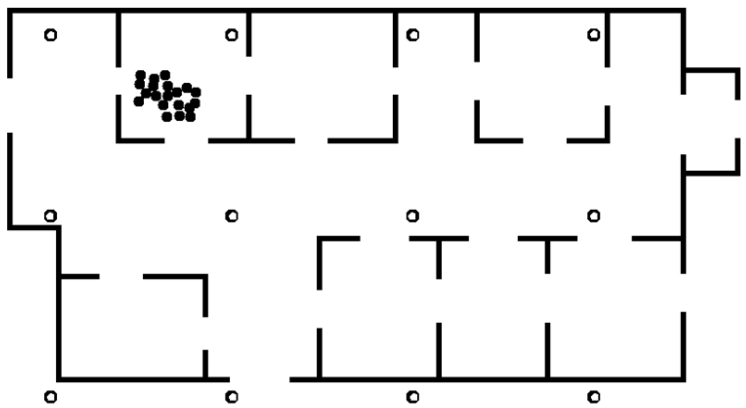


Figure 75: Experiment Building Layout showing initial occupancy (filled circles) and sensor locations (hollow circles)

In these experiments, nine candidate scenarios were generated by directing the crowd to travel to one of nine destination coordinates (e.g. bottom-right room, top-right

exit, top-left exit, top-right room etc.). A tenth scenario was also generated, acting as a stand-in for a real-world evacuation. The simulated sensor readings for this scenario acting as the real readings for the purposes of these experiments. Occupants in the tenth scenario were directed to travel to the bottom-right exterior of the building, which is approximately the same destination as for Candidate Scenario 1.

The sensing rate of the sensors was set to six scans per second, with a trace comparison performed once per second. The experiments covered a 10 second period. These experiments were repeated varying the Range and Accuracy of the motion-detector sensors. Range dictates how close an agent needs to be to the sensor in order of the sensor to detect an occupant (where 100 units is approximate to 3 metres) . Accuracy dictates the chance for the sensor to fail to detect an occupant within its field of vision and range (0.0-1.0 where 1.0 results in all occupants being detected by that sensor).

6.4.4 Pattern Matching Results

The experiments were performed using sensor Range parameters of 80 and 150 units, and sensor Accuracy values of 0.7 and 1.0. For the first experiment, each time a trace comparison is performed the individual sensors record if they have made a scenario match or not, and they note the identifier of the matching scenario. If the identifier is not that of Candidate Scenario 1, we consider the sensor to be mistaken (they have matched to an incorrect scenario).

Figure 76 illustrates the ratio of correct to incorrect scenario matches (averaged over 10 iterations of the experiment) made by individual sensors for each combination of Range and Accuracy values in the experiments; for instance with Range of 80 and accuracy of 0.7, after four seconds we observe an average of approximately six correct matches for every one incorrect match.

Figure 77 illustrates the number of correct and incorrect matches over time, averaged over 10 iterations of the experiment: it was found that the average number of correct matches ranged from 0-4 with the number of incorrect matches ranging from 0-1. We observe that the more sensor comparisons that have been performed, the greater the number of correct scenario matches. We also observe that a relatively small number of sensors (5-6 out of the 12 sensors) tend to identify no particular scenario, this is a result of these sensors having a limited view of the "real" scenario and hence lacking enough observations to consider any particular scenario to be a good match as there is little to distinguish any of them from the observations (as they would not have observed any motion).

The second experiment was performed to determine how well the Base Station could identify a matching scenario when it receives all of the Trace Comparison confidence

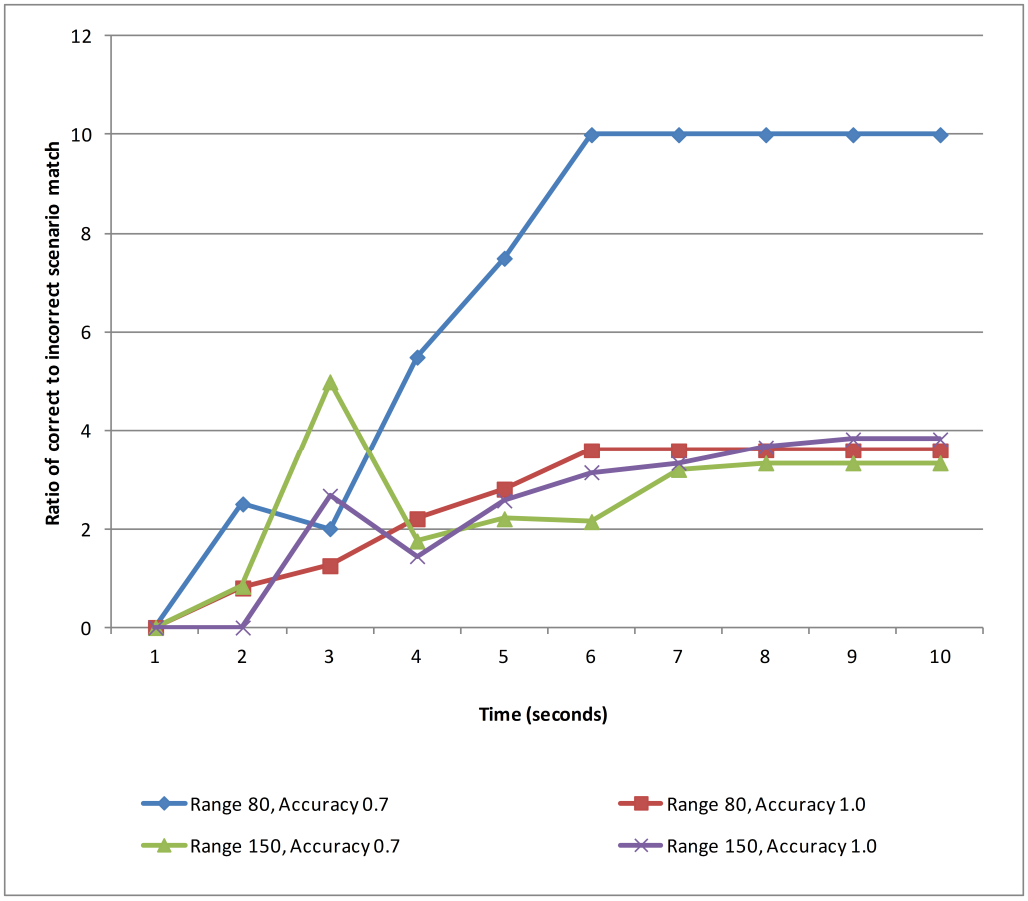


Figure 76: Ratio of correct to incorrect scenario matches at individual sensors (average over 10 iterations)

results as periodic report messages from each sensor node (nodes report the values for all the scenarios, not just their best match). By summing all of these scenario confidence values and dividing by the number of Candidate Scenarios, the Base Station can determine if any particular scenario is a good match.

Figure 78 illustrates the averaged results from each combination of Range and Accuracy values for this experiment. This graph plots the proportion of non-zero matches associated with each scenario as the evacuation progresses. Initially the reports from the sensors paint an ambiguous picture as there is little to distinguish one scenario from another. However, as the crowd begins to move through the building, the average confidence in Scenarios 1 and 8 begins to rise. At approximately the 6-second mark, confidence in Scenario8 begins to fall as the occupants have opted to head to the bottom-left rather than the bottom-right of the building in this candidate scenario. From this point on, the Base Station observes that Scenario1 has at least twice as many non-zero matches with the real data than the next best scenario. By combining the basic match reports from the sensors, the Base Station can determine which scenario is most similar to that which is occurring at an early stage. At this point, the Base Station could reproduce the graphical display of the matching scenario to fire safety personnel, or determine whether the current evacuation plans

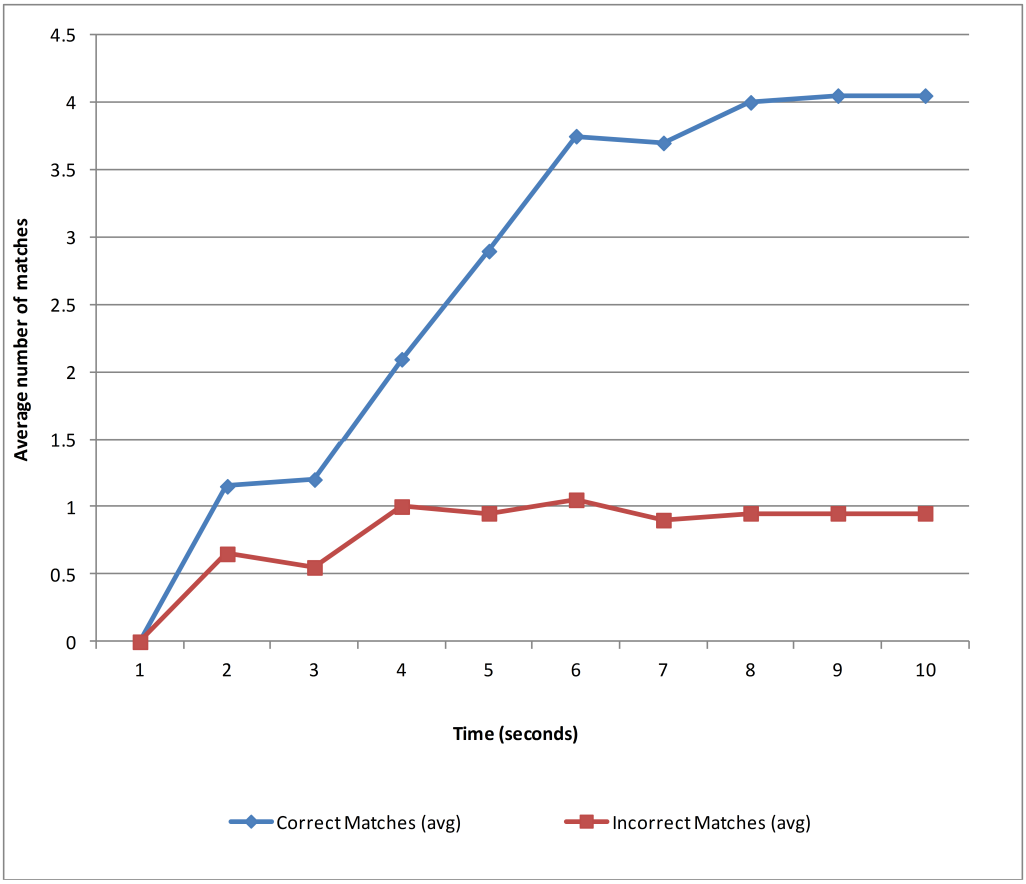


Figure 77: Average number of correct and incorrect scenario matches by individual sensors (average over 10 iterations)

employed in the building should be replaced with others more appropriate for the current scenario.

6.5 Conclusions

By exploiting macroscopic building model analysis to decompose simulation problem space, we were able to achieve highly scalable microscopic simulation and simulate multiple future states of building evacuation which would not otherwise be feasible within the timeframes of emergency response due to the explosion in problem complexity without such decomposition. Simulation of multiple future states allows for contingency simulation and planning; having computed many future states in this manner, the associated simulated sensor readings can be used as the basis for predictive pattern matching during real emergencies. By matching readings to simulation states (scenarios), basic sensing capability can match to rich simulation information. Using basic binary sensing models, we were able to match particular scenarios with confidence; allowing for basic low-complexity sensing to report their readings in terms of complex microscopic detail through simple scenario ID matching.

The network flow graph-based problem decomposition (Section 6.2) computes quickly

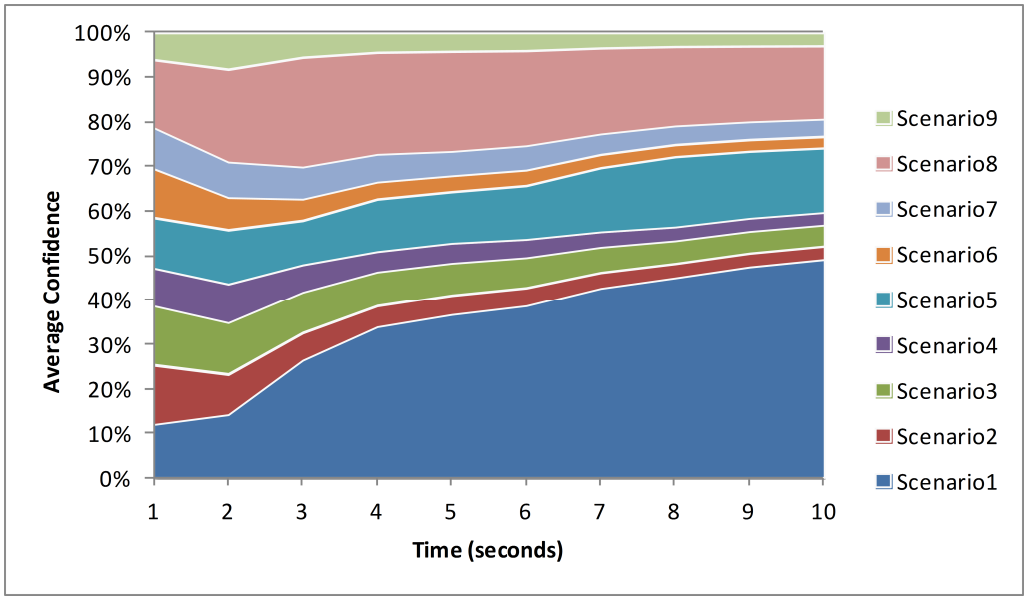


Figure 78: Average % confidence in each scenario from Base Station perspective

and produces sub-problems for simulation that can individually simulate fewer occupants than the overall evacuation scenario, and allow for branching future states without an explosion in the problem size. We developed and implemented a methodology that produces manageable numbers of evacuation simulation instances to compute to simulate multiple potential future states. We demonstrated that this approach can filter out a vast number of future states based on spatial and temporal proximity of occupants evacuating. In our experiments we found that this problem decomposition approach produces a manageable set of future states to simulate, for a typical building and population size. While the individual instances can be computed on a single device, distributed computation resources (e.g. parallel processing or cloud computing) could be an effective resource for exploiting the subdivided problem. Exploring these computing solutions for parallel computation of evacuation simulation problems are a logical next step for exploiting decomposition using flow graphs.

In 6.3 we designed and implemented an emergency monitoring strategy based on prediction using a multi-agent simulation. Experimental results showed that this strategy could be used to achieve a quick response to changes in the progress of an evacuation without requiring terminal nodes to wait for a response from a centralized Base Station. These preliminary results use simplified sensor models and a simple scenario confidence calculation mechanism to identify good scenario matches. We believe that a quick characterisation of the evacuation progress allows nodes to locally make a decision as to the most applicable (pre-computed) evacuation strategy.

7 Conclusions and Future Work

7.1 Conclusions

Integration of micro- and macroscopic models requires that both model types represent the same simulation world. To achieve this we extracted graphs algorithmically from the microscopic building geometry, and with graph simplification and augmentation we created multiple graph hierarchy levels, suitable for different purposes. High-density, high-detail graphs were used for microscopic agent planning, while sparser graphs were used as a basis for Network Flow Graph generation. As these graphs have a common root (derived from the original geometry), routes in one can be converted to the other, allowing for the relaying of plans (such as those generated by flow-based planners) to agents but not requiring the agents to follow those routes precisely, instead allowing them to make use of a more detailed dense graph to follow the list of goals.

Flow-graph coupling to microscopic multi-agent simulation requires that the agent model of pedestrian movement is realistic, otherwise the Flow Graph will not correspond much with real pedestrians. To validate the EvacSim agent model, we compared its results with real-world pedestrian experiments in bottlenecks of various widths and found the model to correspond well with these experiments. In the case of crowds merging, we found that the agent model had to be expanded in order to reflect the behaviour of pedestrians when negotiating merges and after doing so, the model reproduced the relationships between Flow and Density of crowd movement before and during the merging of groups.

By using detailed microscopic social force modelling of pedestrians to establish the parameters of more abstract macroscopic network flow graphs, Flow Graphs wer able reflect some of the greater detail and fidelity of microscopic modelling. While Flow Graphs do still have some shortcomings, particularly in how they model the interaction of occupants at junctions, through coupling with microscopic simulation we achieved high levels of accuracy for traversal time estimates for groups travelling through the building. Using coupled graphs with a dynamic evacuation planner, we showed that congestion-aware evacuation planning with coupled graphs performed well compared to shortest-path or safety-maximising plans.

By taking this approach of coupling macroscopic graphs to macroscopic simulation, we were also able to make use of the reasoning power of graph models to sub-divide the simulation space of microscopic simulation. Subdivision of simulation problems based on the interaction of agents based on time and space allows for lower complexity microscopic simulation, and increases the scope of microscopic simulation to allow for fast, multiple future state simulation. Without macroscopic-level analysis

to reduce the simulation space, the permutations of inter-agent interaction would be infeasible to compute within the timeframes of emergencies. By simulating multiple possible future states of evacuation, we made use of the simulation data to provide future state sensor fingerprints to building sensor networks, allowing them to match readings with simulations during emergency, allowing for rapid response or implementation of contingency plans.

7.2 Future Work

In investigating the impact of crowd merging in flow graphs, we found that the interaction of different flows on different edges was still a factor within the microscopic simulation. Occupants in different parts of graph models may still be near each other within physical space and without modification, the graph perspective of the simulation world cannot detect this. In Chapter 5 we suggested some modifications to graph models which could help capture some of these interactions which would expand on the predictive power and accuracy of graph models. Similiarly, Dynamic Evacuation Planners that make use of Network Flow Graphs for planning also encounter these difficulties; modifying planners to account for these interactions through methods such as limiting the simultaneous departure or arrival of multiple flows at common nodes, or modelling the limited capacity of nodes and edges (instead of just edges as is the case with some planners) could go some way to improving the planners.

In Chapter 6 we described an emergency monitoring strategy based on prediction using the microscopic multi-agent simulation to simulate the future states. Experimental results showed that this strategy could be used to achieve a quick response to changes in the progress of an evacuation without requiring terminal nodes to wait for a response from a centralized resource. These preliminary results used simplistic sensor model and a simple scenario confidence calculation mechanism to identify good scenario matches. Quick characterisation of the evacuation progress allowed nodes to identify matches to pre-simulated scenarios, which in turn could be used to select appropriate evacuation strategies. By implementing more realistic sensor types and investigating more sophisticated scenario matching methodologies at the sensor and base-station, this approach to real time dynamic response could be expanded, for instance using Dynamic Time Warp pattern matching [87] or using heterogeneous sensor types.

In future work we will expand on these experiments, implementing more realistic sensor types and investigating more sophisticated scenario matching methodologies at the sensor and Base Station. When individual sensors were required to match their readings with a candidate scenario, we discovered that a relatively small set of

sensor nodes were able to determine that one of these scenarios was a strong match, as most sensors did not have a clear field of view of the evacuating group. While the accuracy of these matches was generally quite good with a typical correct:incorrect ratio of 4:1, the Predictive Simulation Reporting scheme could be expanded to allow for local sensor nodes to collaborate together in isolating positive scenario matches. Local sharing of match confidence may reduce the number of incorrect matches without introducing substantial additional network traffic.

In Section 2.5 we described some agent grouping approaches that enable individual microscopic agents to amalgamate into groups, approaches which could be used to allow for agent information sharing or group-level planning behaviour. Group formation combined with flow graphs could allow for more powerful agent planning capability, enabling groups to reason about their utilisation of space based on group size to plan accounting for the probable congestion they would experience taking different routes. Navigation graphs (as generated in Chapter 4) can be used by agents to plan routes based on shortest distance using algorithms like A* or Dijkstra's Algorithm, but only account for individual traveral times between nodes. By using macroscopic network flow graphs coupled with the microscopic simulator (Chapter 5) we can augment the agent planning behaviour to account for the limited capacity of edges. While congestion does not have an impact on isolated agents, by expanding the capability of the agent model to allow dynamic group formation, we can introduce the ability for agents and groups of agents to reason about sizes of crowds and the impact crowd size has on traversal times.

In future work we will explore dynamic group formation to allows individual agents in the microscopic simulation to form groups with neighbours, and the sharing of knowledge (such as evacuation routes) and group-level reasoning, similar to the gamma-agent approach described by Olfati-Saber [47]. Group formation would allow not only for groups to chose routes based on the combination of capacity, size and traversal time, but also allow for individual agents to plot alternative routes when they observe queue formation on congested edges on the route they wish to take. By adopting a group formation strategy and integrating Network Flow Graphs with the agent model, the ability for groups of agents to perform path planning accounting for group size and path capacity would great augment the planning capabilities of agents.

In 6.2 we used EvacPlan path allocation as the grouping approach and basis for group routes when generating CrowdRoute objects for problem decomposition. Alternatives to this could make use of agent group formation behaviour as the basis for the initial CrowdRoute generation, or make use of machine learning to determine the likely paths occupants take when traversing space, collected from building sensor networks over day-to-day building use or during emergency evacuation drills.

In investigating the performance of evacuation planning we used a number of path allocation strategies, some of which are more realistic than others for real-world deployment. By allocating paths to occupant agents nearest to the first destination on the path we achieved good evacuation results, but this approach requires a high resolution of localization which may not be available in real world emergency situations. In EvacSim, we localized agents by determining which graph node they were present in; in future we should localize occupants by use of simulated sensor networks, which may provide a very different perspective on the building structure; some approaches may only localise occupants to within a large space, and this may require that many graph nodes be amalgamated into a single node for planning purposes as the fidelity of localization is limited. Integrating a "sensor localization graph" with the navigation and planning graphs described in Chapter 4 would be an interesting expansion to the range of macroscopic models utilized in evacuation simulation and would allow the integration of the sensor fingerprinting approach from Chapter 6 with the agent perspective and the coupled flow graph.

A EvacSim Pedestrian Evacuation Simulator

A.1 Introduction

EvacSim (Figure 79) is a microscopic pedestrian evacuation simulation tool developed in Java, designed for experimental simulation techniques, development of pedestrian simulation models, and ultimately for real-time decision support for evacuation planning and prediction. The goals of EvacSim are to allow for faster-than-realtime simulation of evacuations while using a detailed, high-fidelity simulation model; these two opposing goals result in a simulation design inspired by flocking techniques that keeps update-to-update computation low and maintains simulation accuracy in aggregate by updating often relative to wall-clock time.

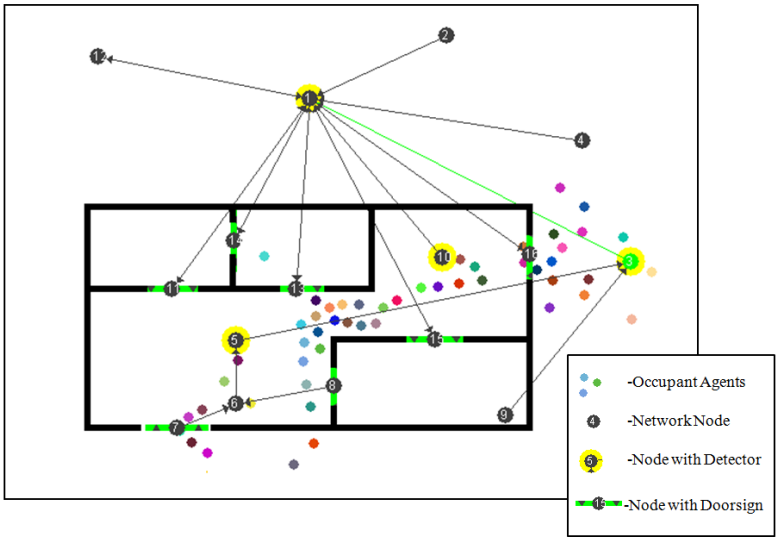


Figure 79: EvacSim Evacaution Simulator with agents and simulated sensor network

In this Appendix, we outline the EvacSim components and describe how the simulation world is represented and generated. This Appendix is ordered in three parts:

- A.2 - EvacSim World Model (Walls, Rooms, graph generation, building models)
- A.3 - Dynamic Elements (simulation ticks, occupant agents, building sensors)
- A.4 - Scalability and Metrics (geospatial indexing, simulation layers, density and velocity collection, heatmaps)

A.2 Simulation World Model

Fundamental to the EvacSim simulation world is the representation of the objects in space. The world is made up of physical obstacles, occupant agents, sensor net-

work components and "room" objects that represent traversable free space. All of these exist in two-dimensional Cartesian Coordinate Space and their locations and dimensions are all given in context of this Coordinate Space. Taking a set of physical obstacles (rectangular walls), we divide the simulation world into discrete traversable spaces (Appendix A.2.4) and from this we produce a building traversability graph, which forms the foundation of the macroscopic models used for navigation (Chapter 4), planning (Chapter 5) and problem space subdivision (Section 6.2). The building's geometry is represented as sets of physical Wall objects, which can be manually drawn (Appendix A.2.7) or imported as XML files generated from Industry Foundation Classes 2.3 definitions (Appendix A.2.6) .

A.2.1 Coordinate System

The EvacSim simulation uses an integer coordinate system to define the position of objects in the simulation, with horizontal axis "x" and vertical access labelled "y". The origin (0,0) is located in the top left corner of the simulation, with the two axes increasing in value to the right and down respectively (negative values are not allowed). This coordinate system is chosen as it corresponds to the graphical representation of screen space and simplifies the graphical display of simulation objects. The position and size of objects is expressed in units of this coordinate system (Figure 80, object height and width is given in terms of points on the grid (i.e. height of "5" occupies a span of 5 points on the coordinate system, and corresponds to 5 pixels in display height at normal magnification). Generally, static objects use whole integer values for Coordinates and dimensions, but some dynamic objects (notably, occupant agents) make use of floating point values which are subsequently rounded to the nearest integer for graphical display.

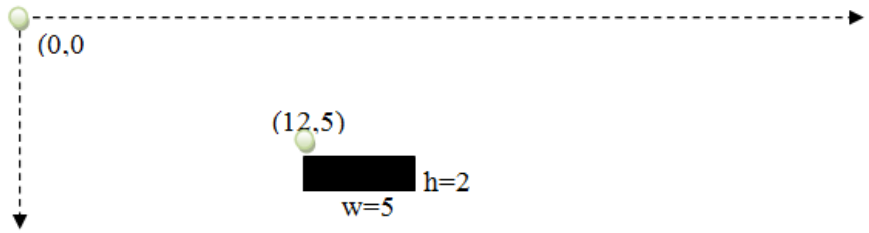


Figure 80: Wall Object in Coordinate Space

As the simulation world is represented in terms of this coordinate space, the correspondence to real world measurements is 15 simulation units to 1 metre. This conversion ratio was chosen in order to display a large amount of simulation space in the simulation display window without image scaling, as simulation units correspond to display pixels in the native resolution. Without scaling, a 1280*800 pixel display corresponds to 4264m^2 Using image scaling, the user can zoom in

and out to show more or less of the simulation world which is rendered internally at the standard resolution (1 uint = 1 pixel) and scaled appropriately using java.awt.Image.getScaledInstance().

A.2.2 Walls

A Wall object is a rectangular object in two-dimensional space. It is defined by its "origin" (coordinate location of the top-left corner of the wall), its "height" and its "width". A building is composed of a collection of Wall objects (Figure 81 which are physically impassable and obscure line-of-sight. Extensions to the Wall object allow for transparent, physical objects (such as windows or guardrails) or for logical subdivision of space without dividing it physically, though these options do not feature in this research.

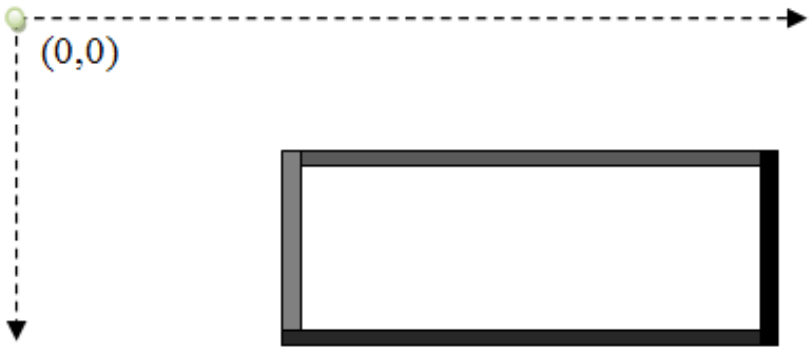


Figure 81: Wall Objects enclosing a space

A.2.3 Line-of-Sight and Collisions

Testing for the line-of-sight between two points is performed by creating a straight line between the two points; The simulator iterates through each Wall in the Building and creates a set of four Lines representing the surfaces of that wall (top, bottom, left side, right side) and performs an intersection calculation between these wall sides and the line-of-sight Line. Any intersections detected represent a blocked line-of-sight (Figure 82).

Collisions between objects are detected by creating a ectangle (java.awt.Rectangle) for each of the objects and checking for an overlap between the objects (via Rectangle.intersection(Rectangle r)). Overlaps with positive area indicate a collision between the objects (Figure 83). In the case of circular objects, the inter-object collision can be detected by simply comparing the distance between the two objects with the sum of their radii. By using circular forms for occupant agents, we can approximate the human shape while availing of this low complexity collision detection

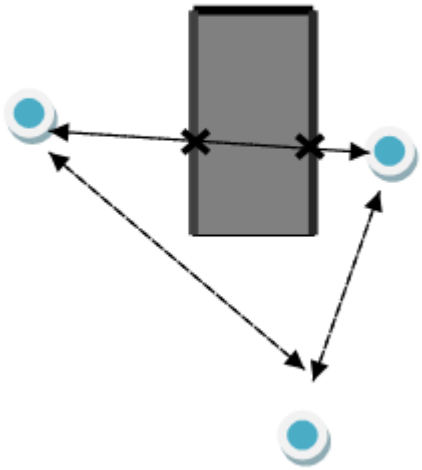


Figure 82: Line of Sight detection

approach. For the case of a Rectangle colliding with a circular object (e.g. agent with a wall), EvacSim uses a Rectangle representation of the bounding box of the circle; this maintains computational simplicity without significant loss of detail.

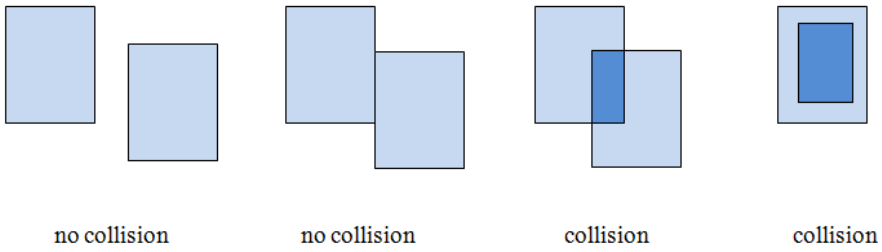


Figure 83: Collision detection (Rectangles)

A.2.4 Room Detection

A Room is a rectangular object similar to a Wall, and has an origin, height and width. EvacSim decomposes a space into a set of Room objects that fully tile the traversable space and are used to produce navigation trees that represent the traversability of the building space (more details on the manipulation and use of navigation graphs are found in Chapters 4 and 5). By using Rectangular Rooms as the space division for EvacSim, identifying Occupant locations is greatly simplified as it is trivial to determine if an object at a particular position lies within a Rectangle (as opposed to within an irregular polygon or other alternative schemes). While this approach can lead to a large number of small rooms when buildings have features like curves, by approximating the curves using many small Wall objects and using graph simplification techniques (loop and triangle removal, described in Chapter 4) we can still achieve good results for building traversability graphs.

Before the the Rooms are generated, EvacSim can first algorithmically detect the building's Doorway spaces and create narrow Room objects for these spaces (in the case that doorways are not included in the building definition with the walls). To discover the Doorway Rooms, EvacSim iterates through each Wall object and finds the two narrow ends (i.e. the two sides of the wall that are narrowest, or all sides in a tie). From each narrow end of a wall, a 1*n Room object is created, where n = the dimension of the wall on that side. This Room object is inflated in the corresponding direction (i.e. to the right from the right side of a Wall, up from the top side and so on) until it collides with another Wall object or Room. In the case of collision with a Room, the Door is discarded. If the collision with a Wall results in an overlap rectangle (Appendix A.2.3) with dimensions equal to 1xn, then the Room is a "Doorway" room and is added to the set of Rooms. The example for generating Doorways from the right side of Walls is given by Algorithm 12 and illustrated in Figure 84. This procedure is repeated for the top, left and bottom of Walls. By using a maximum Doorway size of 40 units, we can ensure that Doorways are only generated for spaces up to 2.66m wide, avoiding the generation of Doorway objects that bridge large gaps between structures such as support columns.

Algorithm 12: Righthand side Doorway generation

```
Data: (Set) Walls
Result: (Set) Doorways
(Set)Doorways = new Set;
(int) sizeLim = 45;
foreach Wall w ∈ Walls do
    if w.width > w.height then
        (Room)right = new Room((w.topRight.x-1,w.topRight.y), w.height,
        1);
        while intersected == null ∧ (right.width < sizeLim) do
            right.inflateRight(1);
            Wall intersected = intersectCheck(right,Walls)
        end
        if intersected ≠ null then
            if (clipped.topLeft == right.topRight)
            ∧ (clipped.bottomLeft == right.bottomRight) then
                Doorways += right
            end
        end
    end
end
```

Having iterated through each Wall object (and identified the available "Doorway" Rooms that can be generated), EvacSim moves on to general Room Detection. A Room can be "inflated", which increases the dimension of the Room in a particular direction, e.g. the Room can be expanded one unit to the right, or up. Similarly the Room can be deflated to reduce a dimension appropriately. Tiling the building

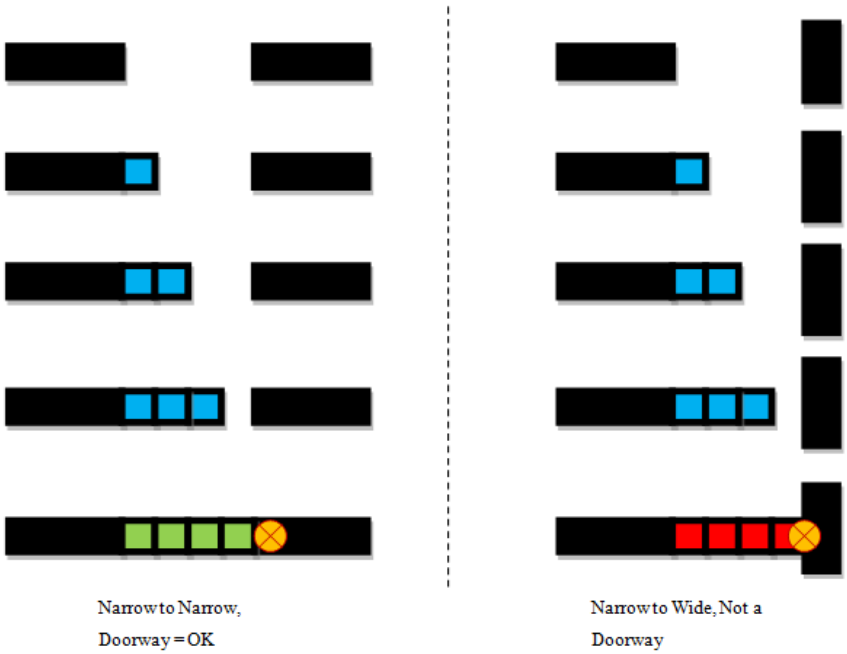


Figure 84: Doorway Detection

space is achieved by creating 1x1 Room objects and inflating these repeatedly to the right, and down, until they collide with a Wall or an existing Room. EvacSim generates a 1*1 Room at the first empty position (i.e. not covered by a Room or Wall) in the simulation world, starting at 0,0 and iterating down, then right. The world bounding space is used to limit the amount of space checked and tiled by the rooms, in this work a 3000 * 3000 bounding space is used. (Algorithm 13) to prevent infinitely expanding Rooms outside of the building.

When collisions are detected, the Room is deflated 1 unit (to pre-collision values) and the inflation in the direction that cause the collision is "locked" (Figure 85). Inflation continues on the unlocked direction until the next collision occurs. Again the direction is deflated 1 unit and the Room is complete. The next empty coordinate is searched and the process continues until the building bounding space is completely covered by Rooms and Walls.

Algorithm 13: Room Generation

```
Data: (Set) Walls, (Set) Doorways
Result: (Set) Rooms
(Set) Obstacles = Walls ∪ Doorways;
(Set) Rooms = new Set;
(int) limx = 3000;
(int) limy = 3000;
(Coordinates) spawnpoint = (0,0);
while spawnpoint.x ≤ limx do
    while spawnpoint.y ≤ limy do
        if !clipped(spawnpoint, Obstacles) then
            (Room) newRoom = newRoom(spawnpoint, 1,1);
            (boolean) LockR = false;
            (boolean)LockD = false;
            while !(LockR ∧ LockD) do
                if !LockR then
                    newRoom.inflateRight(1);
                    if intersects(newRoom,Obstacles) then
                        LockR = true;
                        newRoom.deflateRight(1);
                    end
                end
                if !LockD then
                    newRoom.inflateDown(1);
                    if intersects(newRoom,Obstacles) then
                        LockD = true;
                        newRoom.deflateDown(1);
                    end
                end
            end
            Rooms += newRoom;
            Obstacles += newRoom;
            spawnpoint.y++;
        end
        spawnpoint.y = 0;
        spawnpoint.x++;
    end
end
```

A.2.5 Traversability Graph

The Traversability Graph is generated to represent the traversability between different Rooms in the building. This Graph is composed of Rooms and Intersection objects as graph nodes with a link between any nodes that are adjacent to each other in the building. An Intersection is a special case of Room, representing the space where two Rooms touch each other; a narrow rectangle 1-unit wide or tall which is connected to the two adjacent Rooms (Algorithm 14). Each Room or Intersection

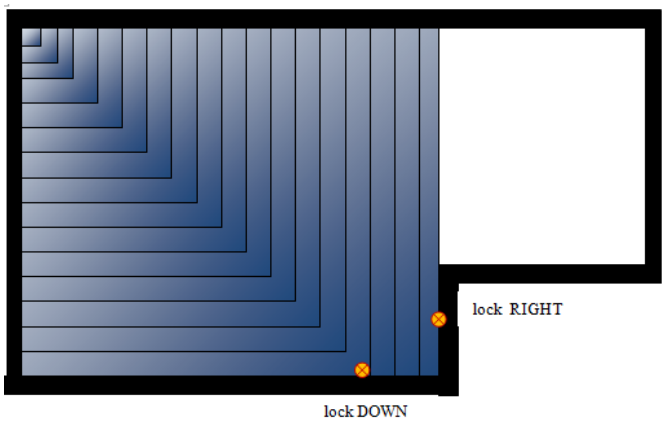


Figure 85: Room generation

is represented in the Graph as a Node with its position at the centre of the Room or Intersection. In generating the Intersection objects, links between Intersections and Rooms are also created (Figure 86).

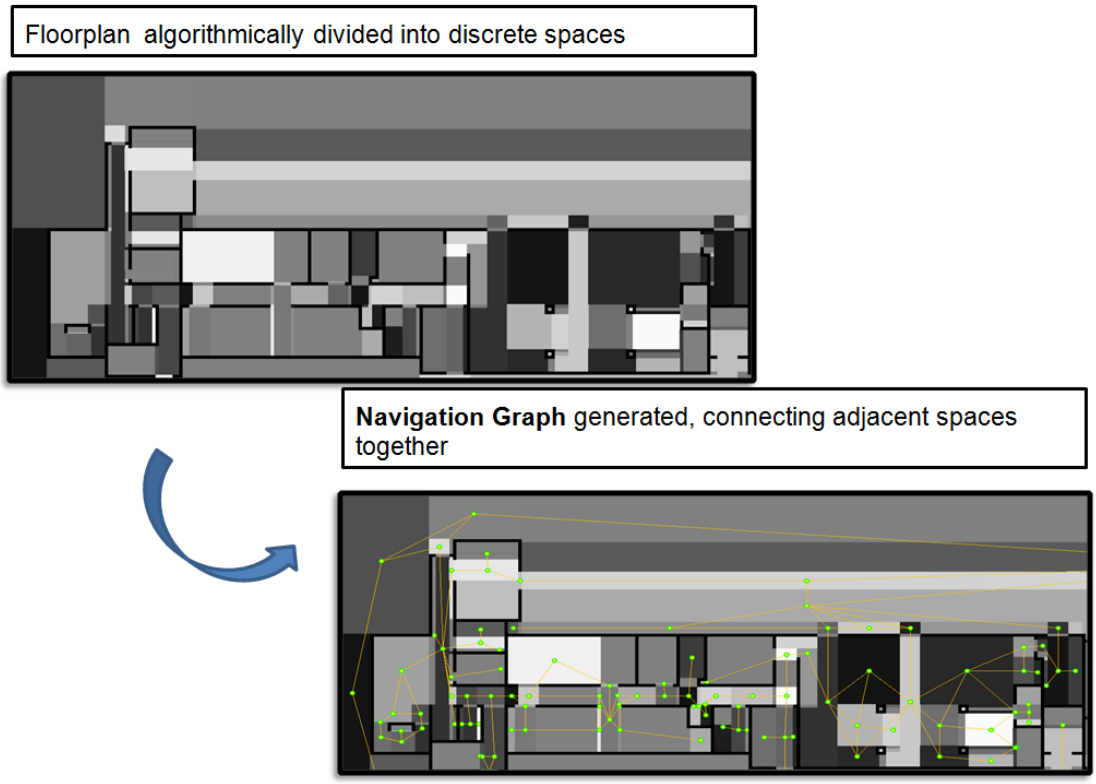


Figure 86: Generation of Traversability graph from Rooms and Intersections

Algorithm 14: Intersection Generation

```
Data: (Set) Rooms, (Set) Doorways, (Graph) g
Result: (Set) Intersection
(Set)Intersections;
(Set) Rooms = Rooms ∪ Doorways;
foreach Room r ∈ Rooms do
    r.inflateRight(1);
    r.inflateDown(1);
end
foreach Room r ∈ Rooms do
    foreach (Room s ≠ r) ∈ Rooms do
        Room intersection = overlap(r,s);
        if intersection.area > 0 ∧ !Intersections.contains(intersection) then
            Intersections.add(intersection);
            Edge e = (r,s);
            g += e;
        end
    end
end
foreach Room r ∈ Rooms do
    r.deflateRight(1);
    r.deflateDown(1);
end
```

Having created the basic navigation graph (Adjacency Links between Rooms and Intersections), additional links can be introduced to create more complex graphs (Figure 87). Examples of additional links are:

- Common Room link (connecting each of a node's neighbours with new links, Algorithm 15)
- Line-of-Sight link (connecting nodes with clear line of sight to each other, Algorithm 16)

Links can be added to the graph to increase the number of navigation options available to agents. More detail on Traversability Graphs and Node/Edge generation options can be found in Chapter 4.

Algorithm 15: Common Room link

```
Data: (Graph) g
foreach Node n ∈ g do
    foreach neighbour Node m of n do
        if !g.connected(m,n) then
            Edge e = (m,n);
            g += e;
        end
    end
end
```

Algorithm 16: Line-of-Sight link

```
Data: (Graph) g
foreach Node n ∈ g do
    foreach Node m ≠ n in g do
        if !g.connected(m,n) ∧ lineOfSight(m,n) then
            Edge e = (m,n);
            g += e;
        end
    end
end
```

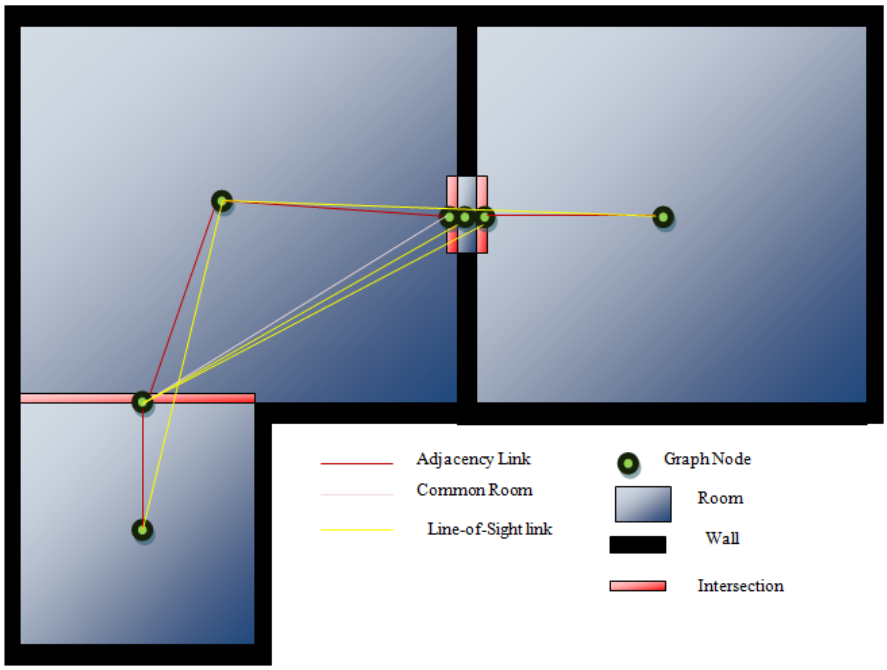


Figure 87: Traversability Graph

Routes between two Rooms (source Room and destination Room) in the navigation graph can be discovered by performing Dijkstra's shortest-path algorithm or the A* algorithm (using straight-line distance between the room centres as heuristic) to determine the shortest route between two points (cf. Section 2.4.2). The Route is returned as a list of Rooms and Intersections which give the sequence of locations that must be traversed in order to travel from the source to the target. To find the path between two arbitrary Coordinate positions in the building, first the Room location of the point must be found (Algorithm 17).

Algorithm 17: whereIsThis method

```
Data: (Set) Rooms, Coordinates c
foreach Room r ∈ Rooms do
    if intersects(r,c) then
        return r;
    end
end
```

Having identified the Room locations of the source and destination (and thus, the graph nodes for each), the path can be discovered using A* or a similar algorithm. This approach simplifies the problem and sometimes leads to suboptimal routes (e.g. while the node-to-node distance may be the shortest, the source and destination Coordinates may be far from the centre of the source and target Room centres). Alternatively, temporary graph Nodes can be generated at the Source and Destination Coordinates and added to the graph at the time that the shortest path algorithm is called.

A.2.6 Building Definition Files

Raw Text Building Definition

The basic Building definition file used by EvacSim is composed of tuples representing Wall origins, heights and widths, delimted by an "ED" symbol. An example Wall with a top-left Corner at (45,50) and a height of 90 and width of 6 would be listed as "45 50 90 6 ED". When predefined Doorways are used, these appear at the endl of the building definition file after a "DOORS" delimiter (Figure 88) and are formatted using the same tuple style as Walls.

```
192 156 6 102 ED 390 156 6 312 ED 696 162 60 6 ED 696 270 66 6 ED 696
378 102 6 ED 564 474 6 138 ED 114 474 6 396 ED 114 396 84 6 ED 120 156
180 6 ED 120 156 6 24 ED 462 456 18 6 ED 462 384 36 6 ED 468 384 6 102
ED 570 306 84 6 ED 570 162 102 6 ED 360 450 30 6 ED 360 336 78 6 ED
282 336 6 78 ED 282 342 72 6 ED 282 444 36 6 ED 120 408 6 36 ED 204
408 6 36 ED 234 372 42 6 ED 234 366 6 48 ED 240 156 54 6 ED 240 234 54
6 ED 210 282 6 36 ED 126 282 6 48 ED 444 162 66 6 ED 444 270 126 6 ED
450 390 6 12 ED 294 156 6 36 ED 0 564 6 840 ED 834 0 570 6 ED 72 396 6
54 ED 72 384 18 6 ED 72 336 18 6 ED 78 336 6 48 ED DOORS 330 156 6 60
ED 144 156 6 48 ED 240 210 24 6 ED 174 282 6 36 ED 72 354 30 6 ED 156
408 6 48 ED 282 414 30 6 ED 360 414 36 6 ED 462 420 36 6 ED 510 474 6
54 ED 444 228 42 6 ED 570 264 42 6 ED 696 336 48 6 ED 696 222 48 6 ED
```

Figure 88: Example building definition

Building definitions can be generated either manually (via a building drawing tool, (Appendix A.2.7) or from an XML input (Appendix A.2.6).

```
<?xml version="1.0" encoding="UTF-8" ?>
<WIDesign floorplan="NIMBUS-GROUND-FLOOR" originx="0" originy="0">
  <FloorPlanWalls>
    <wall id="wall_0" Material="heavy">
      <startpt>
        <x>32.73</x>
        <y>15.67</y>
        <z>0</z>
      </startpt>
      <endpt>
        <x>32.73</x>
        <y>19.82</y>
        <z>0</z>
      </endpt>
    </wall>
  </FloorPlanWalls>
</WIDesign>
```

Figure 89: Example WiDesign XML building definition

Two-dimensional XML Definition

The XML import facility accepts building definitions where Walls are represented by 1-dimensional lines in two-dimensional space (with the building separated into multiple floor objects, as is typical in IFC building definitions (Section 2.3)). An example of an XML building definition with a single wall is shown in Figure 89. XML building definitions for buildings originally represented by IFC file definitions are generated via the WIDesign tool [88] developed by Gibney et al as part of the Nembes project, according to the WIDesign XML format.

Separate XML files represent each individual floor of the building; in this Research we focus on simulation for a single floor at a time (Figure 90), though EvacSim can be expanded to model multiple floors in a two-and-a-half-dimension configuration by allowing agents to transfer between two-dimensional floors which are stacked vertically. The XML formatting for buildings uses one-dimensional lines connecting two points in two-dimensional space, which are converted to two-dimensional rectangles for EvacSim by adding height or width to thicken the lines.

Conversion from the 1-dimensional wall objects in the XML definition to the two-dimensional rectangular objects in EvacSim is achieved by laying the one-dimensional walls over a two-dimensional grid of squares: a “Cover Grid” (Figure 91). This Covergrid is a grid of squares of specific dimension (adjustable by parameter, in this work we use a grid of 5x5 squares). Any squares in the grid that are intersected by IFCWalls are labelled as “filled” and used as the basis for Wall generation. By expanding the one-dimensional IFCWalls into two-dimensional Walls, we ensure that

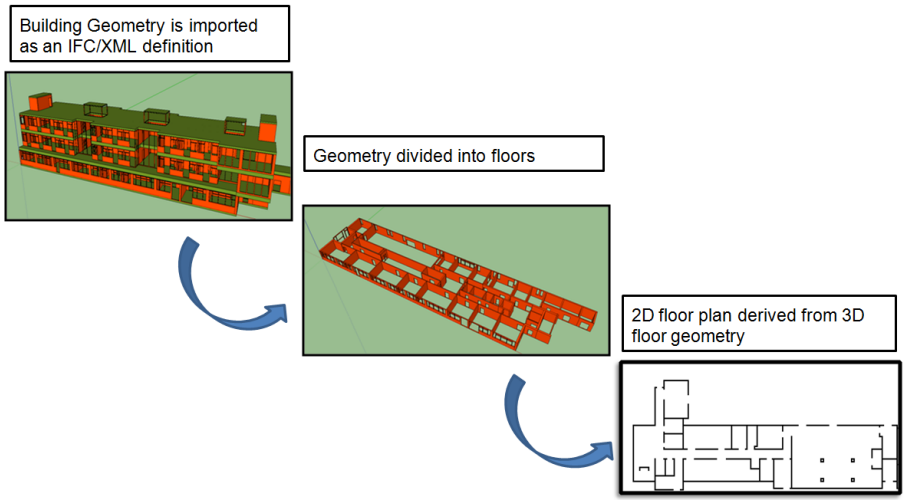


Figure 90: Conversion from IFC to two-dimensional XML representation

agents do not pass through Walls in the case that they travel more than one unit within a single simulation update.

A set of EvacSim Wall objects is created that occupy the "filled" covergrid squares (i.e, each "filled" square becomes a 5x5 square Wall object). This approach leads to a set of many small square walls. While this is sufficient for use in EvacSim, the large number of square Walls interferes with Doorway Detection and will result in a large number of narrow Room objects. To alleviate this, adjacent Wall objects are merged together in straight lines, producing long or tall Wall objects. This procedure has the effect of simplifying the building geometry and adding height and width to the 1-dimensional IFC walls (Figure 92).

A.2.7 Building Drawing Tool

The Building Drawing Tool allows for the manual modification or generation of Building files (using the Raw Text filetype of Appendix A.2.6). As with the XML conversion approach, the Drawing Tool uses a CoverGrid of squares which are filled in by clicking and dragging with the mouse cursor. (Figure 93). A toggle switches between drawing Walls and drawing Doors (in the case that the user requires that specific Doorway Rooms be present before the algorithmic detection of Doorway Rooms described in Appendix A.2.4). The Drawing Tool can load, modify and save raw building definition files, and features basic Clear and Undo functions. The Drawing Tool can also launch directly into EvacSim using the Building it generates.

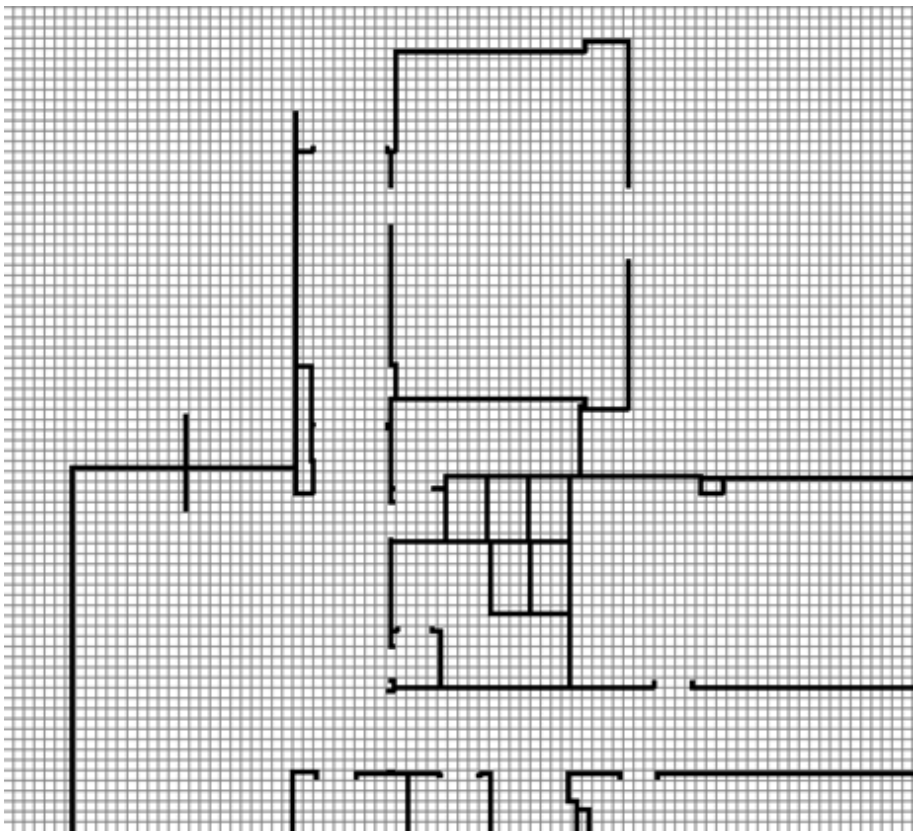


Figure 91: 1-dimensional XMLWalls overlaid on the CoverGrid

A.3 Dynamic Simulation Elements

In this section we describe the dynamic elements of EvacSim. These components are periodically updated according to the simulation clock and are capable of dynamic behaviour and response to changes in the building state. These dynamic components fall into two main categories: Agents and Network components. Agents are used to model occupants in the building, and are capable of motion, planning and observation of their environment. Network components are used to model building Sensor Networks which monitor the progress of the simulated evacuation, or receive actuation instructions to modify building signposting. The sensor network components model sensor networks on a basic level, and are not intended as network simulation testbeds, but rather to model basic limitations of sensor networks and how these can impact on evacuation planning and guidance (limitations such as sensor noise, communication delays and the risks of network disconnection or partition).

A.3.1 Simulation Time and Ticks

EvacSim uses a Main Clock tick loop to control simulation; each tick of the Main Clock calls a tick on the sub-components of the simulation (Agents, Sensors and Network Nodes, Figure 94). The tick() method of an Agent, Sensor or Network Node drives the behaviour of the object, for instance the tick() method of a basic

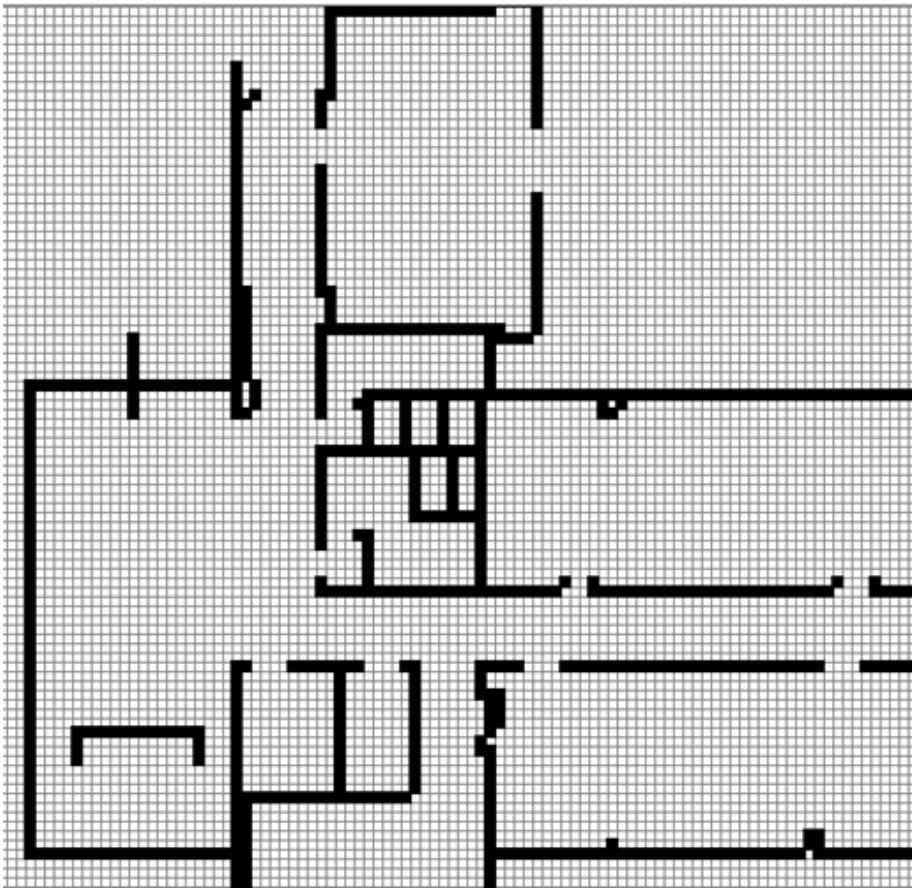


Figure 92: Generation of Walls by filling CoverGrid squares

agent might move its position forward a small amount in the direction the agent is currently facing, or cause the agent to adjust its orientation incrementally over the course of the next few "ticks".

In this research, the relationship between "ticks" and seconds is given by the "TimeUnitAlpha" conversion ratio of 20 ticks per 1 second of wall-clock time; allowing for 20-frames-per-second real-time animation and updating the simulation world at a fast rate. When not tied to the animation refresh rate (20 fps), EvacSim will compute simulation ticks as often as possible; with sufficient computational resources EvacSim will simulate several minutes of simulated time within the space of a few seconds. An optional toggle in EvacSim allows for the computation of many ticks in batches between graphical updates, allowing for accelerated simulation (i.e. graphical updates every 100 ticks instead of each tick), decoupling the simulation speed from the rate of graphical refreshing.

On generation, each agent is allocated a unique integer code-number (generated by incrementing a global agentCodeCounter variable each time a new agent is created). This integer is also used to set agent starting clock values, and is also used as its unique identifier. As each agent has a different starting clock number, periodic agent computations (such as path-planning) are distributed over the agent population rather than all occurring simultaneously (which would cause periodic temporary

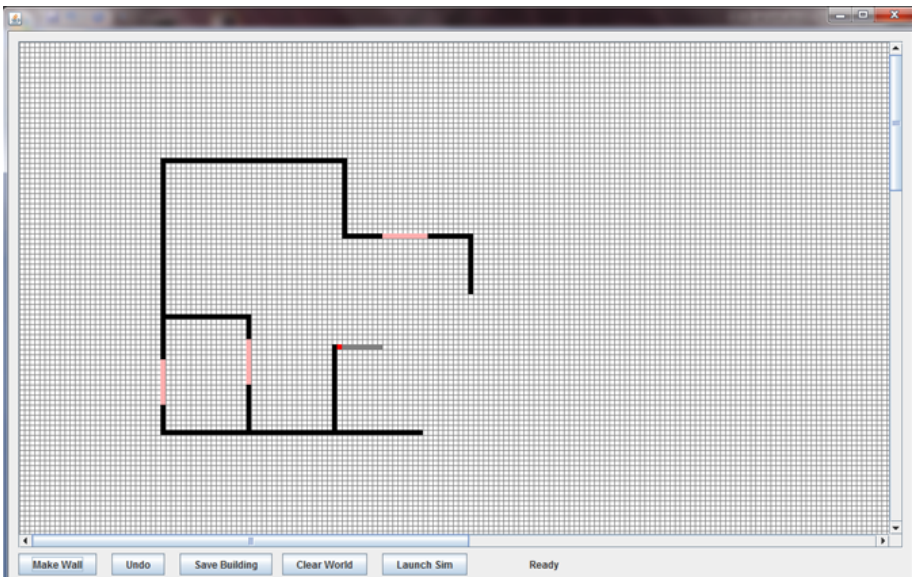


Figure 93: Building Draw Tool

freezes as each agent performs a replan procedure in the same tick). As each agent performs periodic actions such as look-ahead every 50^{th} tick (according to their internal clock), this means that in any given EvacSim tick, no more than $1/50^{th}$ of the population is doing so simultaneously.

A.3.2 Occupant Agents

Fundamental to EvacSim as a pedestrian evacuation simulator is the Occupant Agent. These agents are represented in space as circular objects with a size and a position in two-dimensional Space with floating point precision, which is rounded to the nearest integer when used for graphical display. Using fractional accuracy for agent positions allows for full freedom of movement, without forcing agents to move between positions in whole units.

By manipulating this position, the agent is capable of movement in the Simulation world. The basic agent model uses a "Vector" object which combines the position with a facing direction angle. This angle represents (in degrees) the direction the agent is facing (clockwise relative to the x axis) and is adjusted by the agent to steer and navigate space. Agent movement is conducted by moving the agent a small distance each tick, according to this angle and the agent's movement speed. This Agent can be given a Coordinates "target" and will attempt to steer and move itself towards this target. In this section we give a brief outline of the occupant agents; substantially more detail on the mobility model of the agents is given in Chapter 3.

To move in space, the agent updates its Vector angle is adjusted based each tick and takes a step forwards based on its current speed. Agent Vector adjustment operates

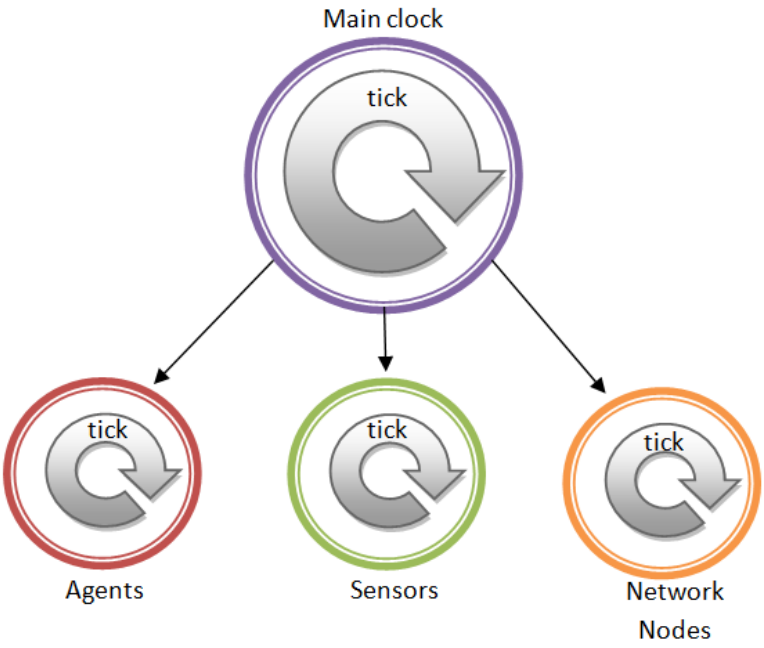


Figure 94: Simulation ticks

as follows:

1. The agent directs the angle towards their goal (a coordinate object)

2. If there are other nearby Agents or Walls in the way, the angle is adjusted to avoid the obstacle

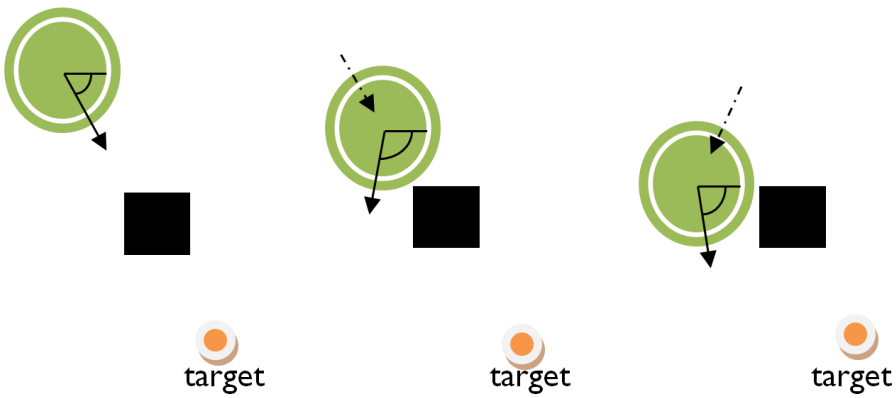


Figure 95: Agent Vector Steering

By adjusting the Vector angle, along with the agent's current speed (the distance moved each time unit), the Agent can manipulate its movement. The agent's current speed is a product of their maximum speed combined with their acceleration and braking factors. Further details on the parameters governing the agent movement are given in Chapter 3.

A.3.3 Collision Correction

Scans for Agent and Wall proximity are performed periodically by checking for collision the agent and any other objects (cf. Appendix A.2.3). Detection of collision between one agent another causes each agent to adjust its position away from the point of collision; this ensures that agents maintain a distance from each other. Collision with Wall objects is discovered by overlaying the bounding box Rectangle of the agent with that of the Wall; to correct this, the agent moves away in the opposite direction of the collision (i.e. to the right if the collision is to the left). The direction of collision is identified by determining which corners of the agent bounding box are within the wall object; if both Left corners of the agent are inside the wall, then a move to the right is used to correct. If only the top left corner is in the Wall, then a move down and to the right is used to correct the collision. If no corners are inside the Wall, then the check is reversed; the location of the wall corners within the Agent is used to determine the direction to move. In the case that no corners of either object lie within the other, then the agent just choose to move arbitrarily to the right; though this event should not occur generally (effectively the agent is inside the wall) unless specified by the user; and is avoided generally by the collision avoidance of agents in simulation.

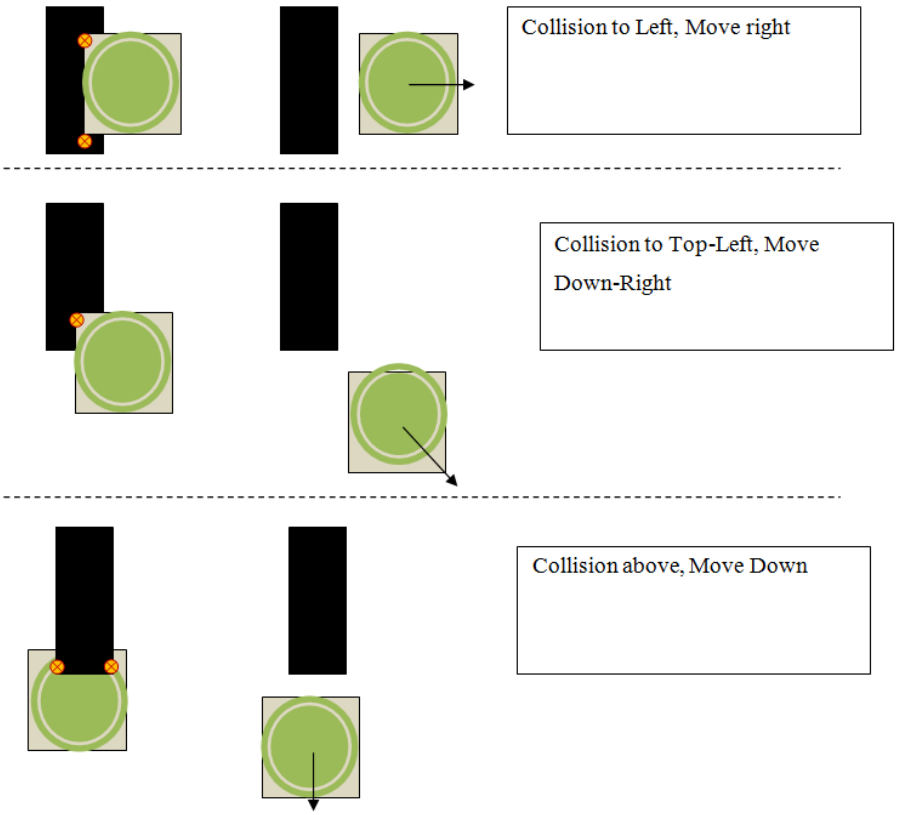


Figure 96: Collision correction examples

A.3.4 Agent Path Planning

Extending the occupant agent to make use of the Traversabilty Graph allows agents to plot routes around the Building. When given a destination coordinates, the Agent can makes a call to the A* method in the Navigation Graph (giving its current Room location as source, and the Room location of the destination as the target destination, by using the WhereIsThis (Algorithm 17) method from A.2.5) and saves the returned sequence of Rooms as a "route" queue. Each tick, the agent moves towards the centre point of the Room which is a the head of the queue (as in the Vector Agent movement). When the agent is within a small distance of this centre point it removes the current Room from the queue and moves onto the next Room in the sequence.

A.3.5 Look-ahead and Re-planning

The Path-planning Agent is extended to allow the agent to look ahead in its route to see if it can take any shortcuts. Periodically (every 50 ticks) the agent iterates over the route in reverse (working backwards from target to source) and performs a Line-of-Sight check on the Nodes. If the agent finds that it has a clear line of sight to a Node during this look-ahead procedure, it skips ahead on the route and makes this Node its next destination.

Occasionally an Agent can be moved off of its route by some external factor, such as crowd movement or an unexpected obstacle. To accommodate this, agents periodically check to see if they are occupying a Room which is not a part of their route. In such an eventuality, the agent calls the Navigation Graph to create a new route to their original destination with the current location as the source.

A.3.6 Sensors and Network Nodes

EvacSim includes a basic simulated Wireless Sensor Network. The purpose of this simulated WSN is to emulate features of WSN such as message delay, sensor gaps/holes, fire damage and out-of-order message delivery. This simulated network capability is intended only as a means to model the limitations of sensing and guidance in emergencies and not as a replacement for dedicated networking simulators such as ns2 [89] or [90].

A.3.7 Network Node and Network

The basic element of the simulated Sensor Network is the Network Node object. Each Network Node has a table of neighbours and a queue of messages. Each tick

of the Network Node causes it to progress through its message queue. Popping a message from the queue sends it on to a neighbour in its table of neighbours (with some percentage chance of dropped packet, which queues a resend). This mechanism ensures that there is a delay between message creation (e.g. a set of sensor readings at a leaf node) and receipt of the message at the sink and that this delay is relative to the number of nodes that the message must traverse.

Depending on the network layout, reliability of links and chance of node failure (e.g. node loss due to fire damage), sensor readings may arrive late, out of order or not at all. Network layout and routing tables are preconfigured by the user, as are node duty cycles.

A.3.8 Extensions to Network Node

The Sensor Node is a special case of Network Node which models the features of a Sensor of some type. There are a variety of Sensor types implemented in EvacSim, the main types are:

- Binary Person Detector - A simple sensor that reports the presence or absence of agents in its vicinity, e.g. a simple motion detector. Sensor reports true or false based on line-of-sight with at least one agent, within a viewing range distance
- Integral Person Detector - A variant of the Person Detector that returns the number of agents in its line of sight
- Boundary Person Detector - A sensor with narrow field of view, the purpose of which is to detect the transition of agents across a boundary, e.g. through doorways. This emulates functionality of a beam or RFID-based sensor and is represented in the simulation world as a rectangle
- Identifying Boundary Detector - Variant of the Boundary Person Detector that can determine the identity of agents it detects. This emulates an RFID-based sensor where an identification can be made based on the RFID reading, e.g. RFID-enhanced ID card

In addition to Sensor Nodes, Actuation Nodes are extensions to the Network Node object that are able to receive instructions to modify their state, which can be reflected in the simulation world. The actuation node implemented in EvacSim is:

- Signpost Node - A Node that is associated with a "Directed Signpost" ([36]) in the simulation world. This Node can receive instructions as to which direction the sign should point, and the sign is visible to agents which can make planning decisions accounting for this signposting if required (such as heading in the direction of the guidance)

A.4 Scalability and Metrics

Real-time or faster-than-real-time simulation requires that EvacSim compute its simulation ticks at a faster rate than the time modelled through those ticks (that is to say, a minute of simulated time should complete in substantially less than one minute). Furthermore, the means to record metrics from simulated evacuations is required in order to evaluate the performance of planners or agent usage of building space. In this section we describe the Layer and Geospatial Indexing features of EvacSim that allow for simultaneous parallel simulations and simulation scalability. We also describe the agent motion recording and playback features of EvacSim, and the recording of density and velocity metrics which can be visually displayed as heatmaps.

A.4.1 Layers

EvacSim can divide elements of the simulation into "layers" with static objects such as Building walls occupying a basic "common layer". Objects within a particular layer can only interact with other objects that also occupy that layer, and with objects in the common layer. The purpose of this layer system is to allow for simultaneous simulation of different evacuation scenarios without creating multiple instances of the Simulation. For example, it is possible to simulate two scenarios simultaneously by having objects relating to Scenario 1 occupy "Layer 1" and objects for Scenario 2 occupy "Layer 2" and proceed with the scenario simulations in parallel. This allows for simulation of different Evacuation plans, or different building floors to occur simultaneously.

A.4.2 Agent Recording

Agent Recording is performed by an AgentRecorder object which periodically (e.g. once per 5 ticks) polls all Agents in the simulation for their current position. The Recorder makes a record of the agent and its position and saves this in its cache. Every 400 ticks the Recorder writes its cache to an XML file on the file system.
Example of an EvacSim XML agent recording (over 2 ticks):

```
<recording period="5">
<agent name="Human2">
     <coords>
          <x>657</x>
          <y>527</y>
     </coords>
     <coords>
          <x>654</x>
          <y>529</y>
```

```
     </coords>
</agent>
<agent name="Human3">
     <coords>
          <x>233</x>
          <y>227</y>
     </coords>
     <coords>
          <x>235</x>
          <y>228</y>
     </coords>
</agent>
</recording>
```

A.4.3 Agent Playback

Playing back a recording in the simulation is accomplished by using a Playback object and a set of Playback Agents. Playback Agents are simple agents (one for every agent in the recording) which take instructions from the Playback object. Periodically (according to the sampling period of the recording file) the Playback agent reads the next location from the recording for each agent and instructs each Playback Agent to move to their next location. The Playback Agents move to the next location at a speed of $\frac{1}{n} \times dist$ per tick where:

n =sampling rate,
$dist$ = straightline distance between current sample and the next

Playback agents do not check for collision as they are governed entirely by the XML recording. They are extensions of the basic agent and as such, other agent types treat them as they would any other agent (i.e. Vector Agents will try to steer around them and will move away from them if there is a collision with perimeter points.

A.4.4 Geospatial Indexing

Geospatial Indexing is a mechanism by which the simulation world is divided into discrete two-dimensional Zones for the purpose of simplifying geospatially-coupled calculations such as collision detection . The Simulation world is divided up into a grid of square Zones, which are instantiated with records of which simulation objects (agents, walls) occupy them. Periodically (every 15 ticks) EvacSim updates the Zone squares as to their contents (as Agent objects are mobile and may change position and hence occupy a different Zone square). As long as the Zones are updated at shorter intervals than the time it would take for an agent to traverse across the

width of a zone, there is no loss of accuracy. The size of the Zones is an adjustable parameter; 150 units is the value used in this work as it produces small Zones while ensuring that agents can not travel through more than two Zones between Zone updates.

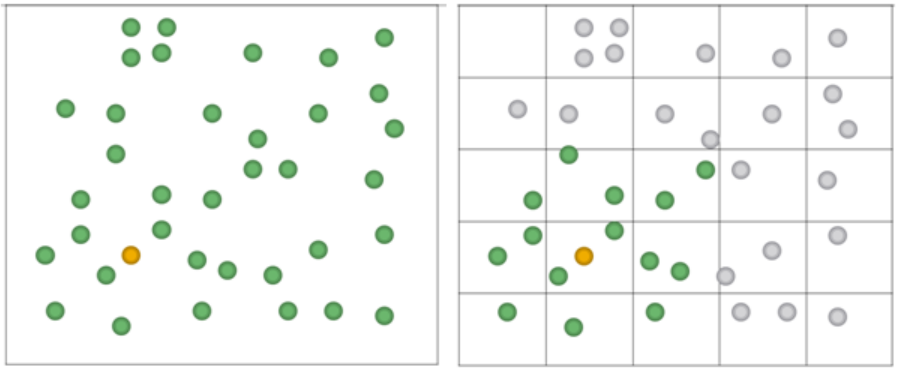


Figure 97: Geospatial Indexing

With these Zone objects, the complexity of collision detection between Agents, and between Agents and Walls is much reduced. It is sufficient for an Agent to limit collision detection scans to those objects which occupy its current zone and the 8 surrounding Zones, in case the agent has moved into a new zone in the time since the last Zone update. In Figure 97 the orange agent would have to check for collisions with 36 other agents, but by using Geospatial Indexing this is reduced to just 14 checks.

As long as the time between Zone updates is shorter than the time it would take for an Agent to traverse between three Zones, this allows for substantial gains on computation time without compromising accuracy (at the expense of a small amount of overhead for maintaining the Zone contents). Smaller Zones improve collision detection but must be updated more often; if an agent is allowed to traverse between zones in between zone updates then correct collision detection cannot be guaranteed. The substantial computation savings Geospatial Indexing allows is shown by recording the average computation time taken per tick when simulating 4000 agents in a 2000 unit square area. Geospatial Indexing shows a large saving in computation time (0.025s per tick on average) , versus the time taken when no indexing is used (0.38s per tick on average). This is compared with the baseline computation time for 10 ticks where no agents are present (<0.1s).

A.4.5 Runtime performance

To evaluate the running time performance of EvacSim, we conducted experiments using timed individual EvacSim instances using a variety of crowd densities, to determine how the number of occupants in a simulation set impacts the running

time of the simulation instance. This experiment recorded the amount of time it took to simulate 40 or 80 seconds of simulated time through a range of population densities, performed on a single core of a Intel Core 2 Duo 3.0Ghz processor using 1Gb of RAM.

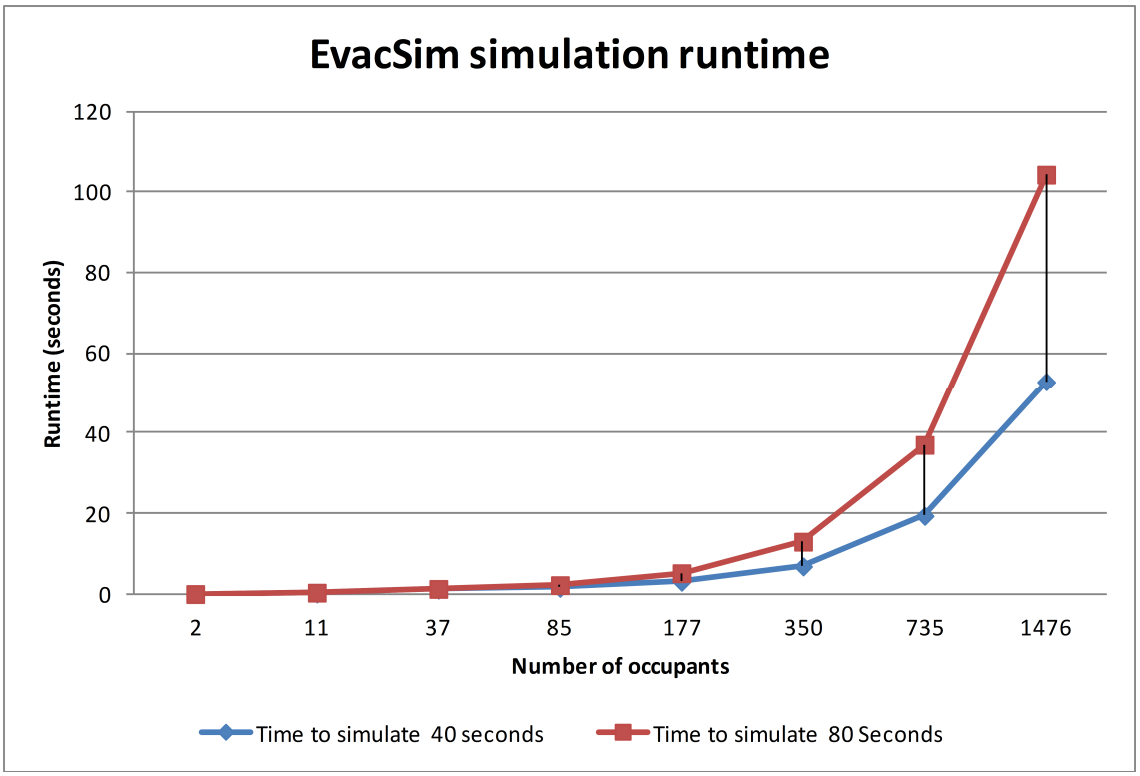


Figure 98: EvacSim running time for 2-1476 agents

At low populations, EvacSim simulates very quickly, each second of simulated time completes in a fraction of wall-clock time. However we find that increasing the population (and thereby, occupant density) the simulation running times quickly become too high to simulate faster than realtime. In Section 6.2 we explore an approach using Flow Graph analysis to decompose the simulation into multiple smaller instances, to reduce the number of occupants in each instance and thereby reduce the number of inter-agent calculations. Decomposition in this manner eliminates inter-agent calculations based on whether agents are likely to meet one another along the paths they follow, which combines with the geospatial indexing (Appendix A.4.4) to reduce complexity.

A.4.6 Simulation Metrics

An important aspect of simulation studies is the collection of metrics, in order to evaluate the performance of the simulator and its models, or to evaluate the operation of evacuation planning algorithms. Evaluation of building space usage and metrics such as crowd density and velocity is possible by covering space with a grid of small square Integral Person Detectors (Section A.3.8). These periodically

count the number of occupants within their square sensing region and also note their velocity. These are used to determine average velocity of occupant agents passing through the sensing region, and the average density of occupants present. Combining these two values we can discover the relationship between Density and Velocity and use this to generate Density/Velocity graphs (e.g. Figure 99, cf. Section 3.6.1 for more details on this graph).

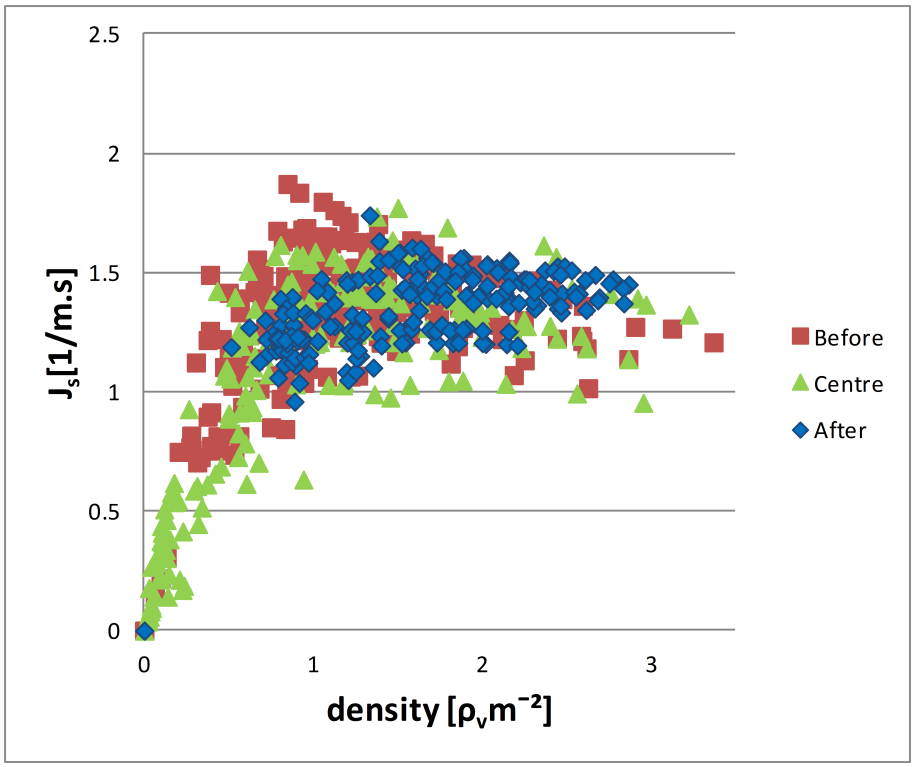


Figure 99: Example Density/Velocity graph plot

A.4.7 Real-time Heatmaps

As the Density and Velocity values are collected by the grid of square Integral Person Detectors, these values can be displayed in EvacSim as a heatmap, a coloured representation of the values in space. By setting the colour of the detector squares based on their density or velocity value (relative to the highest and lowest values among all detector squares) it is possible at a glance to identify how agents are utilising building space, which is extremely useful for identifying potential building problems at the design stage or discovering issues with evacuation plans. Figure 100 shows the Density heatmap generated in real-time using this approach, from Section 3.6.1.

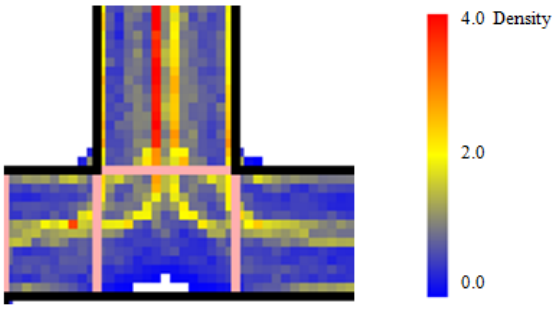


Figure 100: Example real-time density heatmap

A.4.8 High-detail Heatmaps using Voronoi Cells

By collecting a large number of samples over a period of time, a more detailed heatmap can be generated offline through the use of Voronoi Cells [91]. Voronoi cells divide space into individual cells where each point inside of the cell is nearer to the centrepoint of that cell than to the centrepoint of any other cell. To generate these, at each sampling time, the location of each agent is collected and the nearest Integral Person Detector within a grid (Appendix A.4.7) is filled in with that agent's velocity value and that square is labelled as a Voronoi Cell Centrepoint. For each unfilled square, the nearest velocity value of the Centrepoint is identified and used to fill that square (Figure 101).

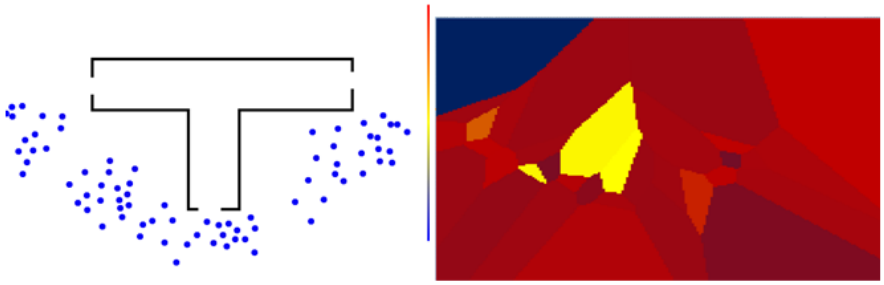


Figure 101: Voronoi Cell velocity Snapshot

For each snapshot, this creates a Voronoi Cell representation of the the simulator world at the time of the snapshot; large cells show a sparse deployment of agents, while the velocity values of the cells represent the speed of the agents. By comparing the area of the cells with their velocity values, the relationship between density and velocity can be retrieved at any given moment. By averaging the velocity values in each grid square over the course of many samples, a rich heatmap can be generated (Figure 102).

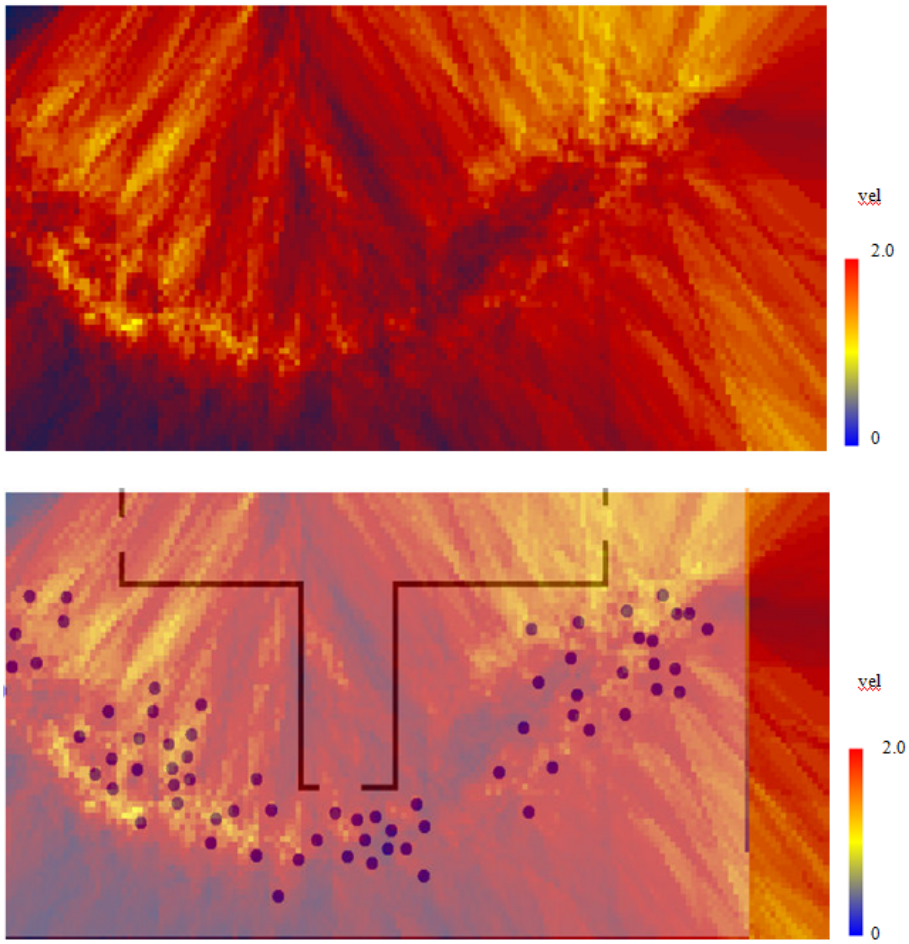


Figure 102: Averaged Velocity, derived from Voronoi Cells

References

[1] J. Zhang, W. Klingsch, A. Schadschneider, and A. Seyfried, "Transitions in pedestrian fundamental diagrams of straight corridors and t-junctions," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2011, no. 06, p. P06004, 2011.

[2] J. Zhang, W. Klingsch, T. Rupprecht, A. Schadschneider, and A. Seyfried, "Empirical study of turning and merging of pedestrian streams in t-junction," *arXiv preprint arXiv:1112.5299*, 2011.

[3] "ped-net.org." http://www.ped-net.org/. Accessed: 27-12-2013.

[4] S. O. Murphy, K. N. Brown, and C. Sreenan, "The evacsim pedestrian evacuation agent model: development and validation," in *Proceedings of the 2013 Summer Computer Simulation Conference*, p. 38, Society for Modeling & Simulation International, 2013.

[5] S. O. Murphy, K. N. Brown, and C. Sreenan, "Problem decomposition for evacuation simulation using network flow," pp. 101–108, 2012.

[6] S. O. Murphy, K. N. Brown, and C. Sreenan, "Predictive simulation & evacuation monitoring in wireless sensor networks," pp. 36–45, 2011.

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[8] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[9] A. H. M. Amin and A. I. Khan, "Parallel pattern recognition using a single-cycle learning approach within wireless sensor networks," in *Parallel and Distributed Computing, Applications and Technologies, 2008. PDCAT 2008. Ninth International Conference on*, pp. 305–308, IEEE, 2008.

[10] S. Goel, A. Passarella, and T. Imielinski, "Using buddies to live longer in a boring world [sensor network protocol]," in *Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on*, pp. 5–pp, IEEE, 2006.

[11] Y. Xu, J. Winter, and W.-C. Lee, "Dual prediction-based reporting for object tracking sensor networks," in *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pp. 154–163, IEEE, 2004.

[12] S. Pattem, S. Poduri, and B. Krishnamachari, "Energy-quality tradeoffs for target tracking in wireless sensor networks," in *Information processing in sensor networks*, pp. 32–46, Springer, 2003.

[13] E. L. Souza, E. F. Nakamura, and H. A. De Oliveira, "On the performance of target tracking algorithms using actual localization systems for wireless sensor networks," in *Proceedings of the 12th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, pp. 418–423, ACM, 2009.

[14] P. Balister, Z. Zheng, S. Kumar, and P. Sinha, "Trap coverage: Allowing coverage holes of bounded diameter in wireless sensor networks," in *INFOCOM 2009, IEEE*, pp. 136–144, IEEE, 2009.

[15] H. O. Sanli and H. Cam, "Joint coverage scheduling and identity management for multiple-target tracking in wireless sensor networks," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pp. 1–6, IEEE, 2008.

[16] S. S. Blackman, "Multiple hypothesis tracking for multiple target tracking," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 19, no. 1, pp. 5–18, 2004.

[17] "Graphisoft archicad." http://www.graphisoft.com/archicad/. Accessed: 27-12-2013.

[18] "Revit for building design and construction." http://www.autodesk.com/products/autodesk-revit-family/overview/. Accessed: 27-12-2013.

[19] V. Bazjanac and D. Crawley, "Industry foundation classes and interoperable commercial software in support of design of energy-efficient buildings," in *Proceedings of Building Simulation99*, vol. 2, pp. 661–667, 1999.

[20] C. E. McDonald, *Framework for a visual energy use system.* PhD thesis, 2013.

[21] T. Froese, M. Fischer, F. Grobler, J. Ritzenthaler, K. Yu, S. Sutherland, S. Staub, B. Akinci, R. Akbas, B. Koo, *et al.*, "Industry foundation classes for project management-a trial implementation," *Electronic Journal of Information Technology in Construction*, vol. 4, pp. 17–36, 1999.

[22] A. Lerner, Y. Chrysanthou, and D. Cohen-Or, "Efficient cells-and-portals partitioning," *Computer Animation and Virtual Worlds*, vol. 17, no. 1, pp. 21–40, 2006.

[23] D. Haumont, O. Debeir, and F. Sillion, "Volumetric cell-and-portal generation," in *Computer Graphics Forum*, vol. 22, pp. 303–312, Wiley Online Library, 2003.

[24] A. Filippoupolitis and E. Gelenbe, "A distributed decision support system for building evacuation," in *Human System Interactions, 2009. HSI'09. 2nd Conference on*, pp. 323–330, IEEE, 2009.

[25] N. Dimakis, A. Filippoupolitis, and E. Gelenbe, "Distributed building evacuation simulator for smart emergency management," *The Computer Journal*, vol. 53, no. 9, pp. 1384–1400, 2010.

[26] A. Filippoupolitis, L. Hey, G. Loukas, E. Gelenbe, and S. Timotheou, "Emergency response simulation using wireless sensor networks," in *Proceedings of the 1st international conference on Ambient media and systems*, p. 21, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[27] "buildingexodus." http://fseg.gre.ac.uk/exodus/. Accessed: 27-12-2013.

[28] N. Chooramun, P. Lawrence, and E. Galea, "Implementing a hybrid space discretisation within an agent based evacuation model," in *Pedestrian and Evacuation Dynamics*, pp. 449–458, Springer, 2011.

[29] J. E. Aronson, "A survey of dynamic network flows," *Annals of Operations Research*, vol. 20, no. 1, pp. 1–66, 1989.

[30] K.-F. Richter, S. Winter, and U.-J. Ruetschi, "Constructing hierarchical representations of indoor spaces," in *Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on*, pp. 686–691, IEEE, 2009.

[31] C. Erikson, D. Manocha, and W. V. Baxter III, "Hlods for faster display of large static and dynamic environments," in *Proceedings of the 2001 symposium on Interactive 3D graphics*, pp. 111–120, ACM, 2001.

[32] A. Sud, E. Andersen, S. Curtis, M. C. Lin, and D. Manocha, "Real-time path planning in dynamic virtual environments using multiagent navigation graphs," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 3, pp. 526–538, 2008.

[33] T. Hadzic, K. N. Brown, and C. J. Sreenan, "Real-time pedestrian evacuation planning during emergency," in *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pp. 597–604, IEEE, 2011.

[34] Q. Lu, B. George, and S. Shekhar, "Capacity constrained routing algorithms for evacuation planning: A summary of results," in *Advances in spatial and temporal databases*, pp. 291–307, Springer, 2005.

[35] T. Tabirca, K. N. Brown, and C. J. Sreenan, "A dynamic model for fire emergency evacuation based on wireless sensor networks," in *Parallel and Distributed Computing, 2009. ISPDC'09. Eighth International Symposium on*, pp. 29–36, IEEE, 2009.

[36] C.-Y. Chen, "A fuzzy-based approach for smart building evacuation modeling," in *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on*, pp. 1200–1203, IEEE, 2009.

[37] A. Ferscha and K. Zia, "Lifebelt: Silent directional guidance for crowd evacuation," in *Wearable Computers, 2009. ISWC'09. International Symposium on*, pp. 19–26, IEEE, 2009.

[38] K. Zia, A. Ferscha, A. Riener, M. Wirz, D. Roggen, K. Kloch, and P. Lukowicz, "Scenario based modeling for very large scale simulations," in *Distributed Simulation and Real Time Applications (DS-RT), 2010 IEEE/ACM 14th International Symposium on*, pp. 103–110, IEEE, 2010.

[39] L. Chittaro and D. Nadalutti, "Presenting evacuation instructions on mobile devices by means of location-aware 3d virtual environments," in *Proceedings of*

the 10th international conference on Human computer interaction with mobile devices and services, pp. 395–398, ACM, 2008.

[40] R. M. Tavares and E. R. Galea, "Numerical optimisation techniques applied to evacuation analysis," in *Pedestrian and Evacuation Dynamics 2008*, pp. 555–561, Springer, 2010.

[41] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[42] A. U. K. Wagoum, M. Chraibi, J. Mehlich, A. Seyfried, and A. Schadschneider, "Efficient and validated simulation of crowds for an evacuation assistant," *Computer Animation and virtual Worlds*, vol. 23, no. 1, pp. 3–15, 2012.

[43] S. Sharma, *Modeling and simulation of multi-agent systems for emergency scenarios*. PhD thesis, Department of Electrical Engineering and Computer Engineering, Wayne State University, Detroit, 12 2006.

[44] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The computer journal*, vol. 7, no. 2, pp. 149–154, 1964.

[45] B. Y. Y. Mohedeen, *Domain Partitioning and software modifications towards the parallelisation of the buildingEXODUS evacuation software*. PhD thesis, University of Greenwich, 2011.

[46] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 25–34, ACM, 1987.

[47] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *Automatic Control, IEEE Transactions on*, vol. 51, no. 3, pp. 401–420, 2006.

[48] A. Borrmann, A. Kneidl, G. Köster, S. Ruzika, and M. Thiemann, "Bidirectional coupling of macroscopic and microscopic pedestrian evacuation models," *Safety Science*, vol. 50, no. 8, pp. 1695–1703, 2012.

[49] A. Schadschneider and A. Seyfried, "Validation of ca models of pedestrian dynamics with fundamental diagrams," *Cybernetics and Systems: An International Journal*, vol. 40, no. 5, pp. 367–389, 2009.

[50] H. Klüpfel, *A cellular automaton model for crowd movement and egress simulation*. GRIN Verlag, 2012.

[51] L. Ji, Y. Qian, J. Zeng, M. Wang, D. Xu, Y. Yan, and S. Feng, "Simulation of evacuation characteristics using a 2-dimensional cellular automata model for pedestrian dynamics," *Journal of Applied Mathematics*, vol. 2013, 2013.

[52] I. G. Georgoudas, P. Kyriakos, G. C. Sirakoulis, and I. T. Andreadis, "An fpga implemented cellular automaton crowd evacuation model inspired by the electrostatic-induced potential fields," *Microprocessors and Microsystems*, vol. 34, no. 7, pp. 285–300, 2010.

[53] Q. Zhang, B. Han, and D. Li, "Modeling and simulation of passenger alighting and boarding movement in beijing metro stations," *Transportation Research Part C: Emerging Technologies*, vol. 16, no. 5, pp. 635–649, 2008.

[54] N. Pelechano, K. O'Brien, B. Silverman, and N. Badler, "Crowd simulation incorporating agent psychological models, roles and communication," tech. rep., DTIC Document, 2005.

[55] N. Pelechano Gómez, A. Malkawi, *et al.*, "Comparison of crowd simulation for building evacuation and an alternative approach," 2012.

[56] N. Pelechano Gómez, K. O'Brien, B. G. Silverman, N. Badler, *et al.*, "Crowd simulation incorporating agent psychological models, roles and communication," 2010.

[57] T. Korhonen, S. Hostikka, S. Heliövaara, H. Ehtamo, and K. Matikainen, "Integration of an agent based evacuation simulation and the state-of-the-art fire simulation," in *Proceedings of the 7th Asia-Oceania Symposium on Fire Science & Technology*, pp. 20–22, 2007.

[58] "Fire dynamics simulator with evacuation." http://www.vtt.fi/proj/fdsevac/?lang=en. Accessed: 27-12-2013.

[59] S. Heliövaara, H. Ehtamo, D. Helbing, and T. Korhonen, "Patient and impatient pedestrians in a spatial game for egress congestion," *Physical Review E*, vol. 87, no. 1, p. 012802, 2013.

[60] S. Heliövaara, T. Korhonen, S. Hostikka, and H. Ehtamo, "Counterflow model for agent-based simulation of crowd dynamics," *Building and Environment*, vol. 48, pp. 89–100, 2012.

[61] C. B. Niederberger, *and Real-Time Simulation for Autonomous Agents using Hierarchies and Level-of-Detail*. PhD thesis, Swiss Federal Institute of Technology, 2005.

[62] T. Becker, C. Nagel, and T. H. Kolbe, "Supporting contexts for indoor navigation using a multilayered space model," in *Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on*, pp. 680–685, IEEE, 2009.

[63] L. Navarro, F. Flacher, and V. Corruble, "Dynamic level of detail for large scale agent-based urban simulations," in *The 10th International Conference on Au-*

tonomous Agents and Multiagent Systems-Volume 2, pp. 701–708, International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[64] L. Navarro, V. Corruble, F. Flacher, and J.-D. Zucker, "A flexible approach to multi-level agent-based simulation with the mesoscopic representation," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 159–166, International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[65] S. Paris, A. Gerdelan, and C. OSullivan, "Ca-lod: Collision avoidance level of detail for scalable, controllable crowds," in *Motion in Games*, pp. 13–28, Springer, 2009.

[66] A. Kneidl, D. Hartmann, and A. Borrmann, "A hybrid multi-scale approach for simulation of pedestrian dynamics," *Transportation Research Part C: Emerging Technologies*, 2013.

[67] M. Sung, M. Gleicher, and S. Chenney, "Scalable behaviors for crowd simulation," in *Computer Graphics Forum*, vol. 23, pp. 519–528, Wiley Online Library, 2004.

[68] T. N. A. Nguyen, J. D. Zucker, H. D. Nguyen, A. Drougou, and D. A. Vo, "A hybrid macro-micro pedestrians evacuation model to speed up simulation in road networks," in *Advanced Agent Technology*, pp. 371–383, Springer, 2012.

[69] D. Helbing, A. Johansson, and H. Z. Al-Abideen, "Dynamics of crowd disasters: An empirical study," *Physical review E*, vol. 75, no. 4, p. 046109, 2007.

[70] G. Proulx and I. M. Reid, "Occupant behavior and evacuation during the chicago cook county administration building fire," *Journal of fire protection engineering*, vol. 16, no. 4, pp. 283–309, 2006.

[71] A. Seyfried, O. Passon, B. Steffen, M. Boltes, T. Rupprecht, and W. Klingsch, "New insights into pedestrian flow through bottlenecks," *Transportation Science*, vol. 43, no. 3, pp. 395–406, 2009.

[72] T. Kretz, A. Grünebohm, and M. Schreckenberg, "Experimental study of pedestrian flow through a bottleneck," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2006, no. 10, p. P10014, 2006.

[73] K. Müller, *Zur Gestaltung und Bemessung von Fluchtwegen für die Evakuierung von Personen aus Bauwerken auf der Grundlage von Modellversuchen.* PhD thesis, 1981.

[74] R. Nagai, M. Fukamachi, and T. Nagatani, "Evacuation of crawlers and walkers from corridor through an exit," *Physica A: Statistical Mechanics and its Applications*, vol. 367, pp. 449–460, 2006.

[75] H. W. Fischer, G. F. Stine, B. L. Stoker, M. L. Trowbridge, and E. M. Drain, "Evacuation behaviour: why do some evacuate, while others do not? a case study of the ephrata, pennsylvania (usa) evacuation," *Disaster Prevention and Management*, vol. 4, no. 4, pp. 30–36, 1995.

[76] A. Sekizawa, M. Ebihara, H. Notake, K. Kubota, M. Nakano, Y. Ohmiya, and H. Kaneko, "Occupants' behaviour in response to the high-rise apartments fire in hiroshima city," *Fire and Materials*, vol. 23, no. 6, pp. 297–303, 1999.

[77] G. Chu and J. Sun, "Decision analysis on fire safety design based on evaluating building fire risk to life," *Safety Science*, vol. 46, no. 7, pp. 1125–1136, 2008.

[78] K. Himoto and T. Tanaka, "Development and validation of a physics-based urban fire spread model," *Fire Safety Journal*, vol. 43, no. 7, pp. 477–494, 2008.

[79] R. Bukowski and C. Sequin, "Interactive simulation of fire in virtual building environments," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 35–44, ACM Press/Addison-Wesley Publishing Co., 1997.

[80] "Fire dynamics simulator and smokeview." https://code.google.com/p/fds-smv/. Accessed: 27-12-2013.

[81] S.-H. Koo, J. Fraser-Mitchell, and S. Welch, "Sensor-steered fire simulation," *Fire Safety Journal*, vol. 45, no. 3, pp. 193–205, 2010.

[82] S. Hoogendoorn and P. HL Bovy, "Simulation of pedestrian flows by optimal control and differential games," *Optimal Control Applications and Methods*, vol. 24, no. 3, pp. 153–172, 2003.

[83] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic," *Journal de Physique I*, vol. 2, no. 12, pp. 2221–2229, 1992.

[84] *Technical Guidance Document L. PartB (Fire Safety).* Department of Environment, Community and Local Government, Ireland, 2006.

[85] *Americans with Disabilities Act (ADA) Accessibility Guidelines for Buildings and Facilities.* Architectural, U. S., and Transportation Barriers, Federal Register 56, 1991.

[86] F. Yuan and L. D. Han, "A multi-objective optimization approach for evacuation planning," *Procedia Engineering*, vol. 3, pp. 217–227, 2010.

[87] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and information systems*, vol. 7, no. 3, pp. 358–386, 2005.

[88] A. McGibney, A. Gurnard, and D. Pesch, "Wi-design: A modelling and optimization tool for wireless embedded systems in buildings," in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*, pp. 640–648, IEEE, 2011.

[89] N. Simulator, "ns-2," 1989.

[90] O. Modeler, "Opnet technologies inc," 2009.

[91] G. Voronoï, "Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs.," *Journal für die reine und angewandte Mathematik*, vol. 134, pp. 198–287, 1908.