Imperial College London
Department of Electrical and Electronic Engineering

A Modelling Approach To Human Navigation in Constrained Spaces

Antoine Desmet

January 2014

Supervised by Prof. Erol Gelenbe

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Electrical and Electronic Engineering of Imperial College London
and the Diploma of Imperial College London

Abstract

In this thesis, we consider algorithms and systems which dynamically guide evacuees towards exits during an emergency to minimise building evacuation time. We observe that the "shortest safe path" routing approach is inadequate when congestion is a predominant factor, and therefore focus on systems which manage congestion.
We first implement a "Reactive" metric which compares paths based on real-time transit times. We find that regular route corrections must be issued to address the constant changes in path delays, and that routes oscillate. We also implement a model-based "Proactive" metric which forecasts the increase in future congestion that results from every routing decision, allowing the routing algorithm to operate offline.
We combine both metrics with the Cognitive Packet Network (CPN), a distributed self-aware routing algorithm which uses neural networks to efficiently explore the building graph. We also present the first thorough sensitivity analysis on CPN's parameters, and use this to tune CPN for optimal performance.
We then compare the proactive and reactive approaches through simulation and find both approaches reduce building evacuation times – especially when evacuees are not evenly distributed in the building. We also find major differences between the Proactive and Reactive approach, in terms of stability, flexibility, sensory requirements, etc.
Finally, we consider guiding evacuees using dynamic exit signs, whose pointing direction can be controlled. Dynamic signs can readily be used with Reactive routing, but since Proactive routing issues routes on an individual basis, one display is required for each evacuee. This is incompatible with dynamic signs; therefore we propose a novel algorithm which controls the dynamic signs according to the Proactive algorithm's output. We simulate both systems, compare their performance, and review their practical limitations. For both approaches, we find that updating the sign's display more often improves performance, but this may reduce evacuee compliance and make the system inefficient in real-life conditions.

Acknowledgements

First of all, I would like to thank my supervisor Professor Erol Gelenbe for giving me the chance to study at Imperial College under his supervision. His guidance, innovative ideas and constructive criticism have shaped this work. While I was at times overwhelmed by the speed at which he provided feedback on my work (and the amount thereof), I am hugely appreciative of the level of support and dedication he displays towards his PhD students. Prof. Gelenbe is also a figure I look up to, for his success in research and career achievements: I value the opportunity I have been given to spend three years under his supervision, which has taught me well beyond the purely technical aspects of research. I would also like to thank him for his financial support, which allowed me to attend several conferences and support for my tuition fees.
I also wish to thank my examiners, Prof Szczerbicka and Dr Demiris for reviewing my thesis, their engaging questions and feedback during my defense.

I must also acknowledge JOY Mining Machinery. I am grateful to those which have seen the benefit in letting a new employee take leave from the company for over three years to pursue further education aspirations and decided to give me a bursary. Amongst others, I would particularly like to thank Manie, Rudie and Peter for their initial support, Kobus and Helen for they day-to-day assistance, and Gustavo for helping me keep up with the business' evolution while I was away.

In many ways, my friends have also contributed to my success. I would like to thank Christina and Omer for their "insider tips" which helped me navigate those three years and steer clear of pitfalls. Special thanks to Gokce for all the engaging talks, his interest into my research – even to the point of spotting the gaps in my ideas. I hope to have duly passed down all their knowledge to the next PhD candidates, and will keep fond memories of working, chatting and laughing with Huibo, Mihai and Mihaelo. Joining the Wakeboarding club at Imperial was also a great opportunity to make friends outside of "the office" and momentarily disconnect from my studies. I'll fondly remember our trips (and their organisation), the laughs, banter, introduction to the London nightlife, and the opportunity to keep a foot in the student lifestyle !

I would also like to thank my parents for their support. My mother, for her love and for looking after me with such dedication, and for her involvement in maintaining my vital stockpile of premium French cheese. I would also like to thank my father for making me press the *send* button on the PhD applications I was so hesitant to send. The fact that he started most new conversations with "*haven't you finished your PhD yet?*" has certainly helped keeping me committed and on track. I credit his great ability to reason in an abstract manner and statistics skills to the clarity and thoroughness of some of the investigations presented in this thesis, and thank him for the time he spent helping me.

Finally, I would like to thank my wife Clare. I acknowledge the huge commitment she made by leaving Australia and following me to the land of long and cold winters and pebbly beaches. She has selflessly encouraged me to follow my dream, knowing that it would make some aspects of her life harder: finding employment overseas, not to mention the threat of vitamin D deficiency! Thank you for being so supportive and understanding, even at the toughest times where my degree was taking most of my free time; but I'm also grateful for all the times she recognised it was time to "pull me out of it" and take me for a walk outside to clear my mind. I also thank her for her dedication in helping me proof-read my thesis, and her patience and interest whilst I was putting the clarity of my research argumentations past her approval.

Contents

List of Tables

List of Figures

Symbols and Acronyms

Δt_R	Display update period of dynamic signs driven by the Reactive algorithm, page 94
Δt_S	Time-step of the Proactive algorithm, page 72
$\delta t_{n\to e,k}$	Period of time where node n's dynamic sign will point towards the edge e within the k^{th} time-step, page 108
ε_S	Deviation in bottleneck load balance (from the Proactive routing algorithm's intended ratio) contributed by the dynamic signs, page 119
$\varepsilon_{\lambda_n}(t)$	Deviation of the instantaneous arrival rate $\lambda_n(t)$ from its arithmetic mean $\overline{\lambda_n}(k)$, page 108
$\varepsilon_\pi(t)$	Deviation of the instantaneous routing probability $\pi_{n\to e}(t)$ from its arithmetic mean $\overline{\pi}_{n\to e}(k)$, page 106
$\varepsilon_{n\to e,k}$	Number of evacuees at node n over- or under-assigned to edge e in time-step k, page 115
$\lambda_n(t)$	Instantaneous arrival rate into node n, page 108
$\overline{\lambda_n}(k)$	Mean arrival rate into node n over the k^{th} time-step, page 108
$\pi_{n\to e}(f)$	Probability of user f leaving node n via edge e, page 105
$\pi_{n\to e}(t)$	Probability that any evacuee leaving node n at instant t is directed towards e, page 114
$\overline{\pi}_{n\to e}(k)$	Mean routing probability from node n towards edge e over the span of the k^{th} time-step, page 107
C_v	Capacity of edge v, i.e. upper limit of $\lvert F(v,t)\rvert$, page 71
F	Population of evacuees, page 71
$F^*_{v,t}$	Set of evacuees queueing for v at the instant t, page 71

$F^B_{v,k}$	Set of evacuees which have made a reservation to the k^{th} time-bin of edge v, page 72
$F_{n\to e,k}$	Set of of evacuees departing node n during the k^{th} time-step, to go on edge e, page 107
$F_{n\to e,t}$	Set of of evacuees departing node n at instant t to go on edge e, page 106
$F_{n,k}$	Set of of evacuees departing node n during the k^{th} time-step, page 107
$F_{n,t}$	Set of of evacuees departing node n at instant t, page 106
$F_{v,k}$	Set of evacuees which have transited through edge v within time-step k, page 72
K	Total number of time-steps, page 72
k	Refers to the k^{th} time-step, page 72
N_s	Set of nodes where signs that assign evacuees to staircases are located, page 96
P	Path composed of an ordered series of vertices $V^P = \{v^P_1, v^P_2, ..., v^P_n\}$, page 71
$p^+(i,j)$	Probability of sending a positive spike from i to j, page 49
$p^-(i,j)$	Probability of sending a negative spike from i to j, page 49
p_r	Probability that a path update will be transmitted to the relevant evacuee, page 86
Q_P	Measure of a path P's quality, page 56
r_R	Ratio of evacuees allocated to a bottleneck, according to the Proactive routing algorithm's plans, page 119
r_T	Ratio of evacuees *effectively* allocated to a bottleneck during the evacuation, page 119
$T_P(F)$	Steady-state time needed for F evacuees to walk through path P, page 23
$T_P(t)$	Time needed, in transient state, for an evacuee departing at time t to walk along the path P, page 71
T_v	Time needed to walk along edge v in free-flow conditions, page 71

w_{ij}^{+}	Neuron i's positive spiking weight towards neuron j, page 50
w_{ji}^{-}	Neuron i's negative spiking weight towards neuron j, page 50
CPN	Cognitive Packet Network: self-aware routing algorithm, page 47
DBES	Distributed Building Evacuation Simulator, page 38
Free-flow	Regime in which evacuees can walk at their nominal speed, i.e. there is no queues or congestion, page 22
Lead Time	Minimum time needed for an evacuee to walk from an endpoint of a path to the other, without obstruction, page 23
RNN	Random Neural Network, page 49
SAN	Self Aware Network, page 34
Saturated	Said of an edge whose maximum allowable flow is reached, page 22
SP	Smart Packet: exploratory packet in CPN routing algorithm, page 48
SP Batch	Process equivalent to sending one SP from each node in the graph, page 56

Preface

Statement of Originality

This thesis is submitted for the degree of Doctor in Philosophy in the Department of Electrical and Electronic Engineering at Imperial College London. I certify that all material in this thesis which is not my own is acknowledged.

Copyright Declaration

Publications

- A. Desmet and E. Gelenbe. Capacity based evacuation with dynamic exit signs. In *Proceedings of the 4th International Workshop on Pervasive Networks for Emergency Management (PerNEM'14)*, March 2014

- A. Desmet and E. Gelenbe. Reactive and proactive congestion management for emergency building evacuation. In *38th Annual IEEE Conference on Local Computer Networks (LCN'13)*, October 2013.

- H. Bi, A. Desmet, and E. Gelenbe. Routing emergency evacuees with cognitive packet networks. In E. Gelenbe and R. Lent, editors, *Information Sciences and Systems 2013 – Proceedings of the 28th International Symposium on Computer and Information Sciences (ISCIS'13)*, volume 264 of *Lecture Notes in Electrical Engineering*, pages 295 – 303. Springer, October 2013.

- A. Desmet and E. Gelenbe. Graph and analytical models for emergency evacuation. In *Future Internet*, 5(1):46 - 55, 2013.

- A. Desmet and E. Gelenbe. Graph and analytical models for emergency evacuation. In *Proceedings of the 2013 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 523 - 527, March 2013.

- A. Desmet and E. Gelenbe. Interoperating infrastructures in emergencies. In E. Gelenbe and R. Lent, editors, *Computer and Information Sciences III*, pages 123 – 130. Springer London, 2013.

- A. Desmet and E. Gelenbe. Identifying critical sub-systems in the simulation of cyber-physical systems. In *Proceedings of the 6th UKSim/AMSS European Symposium on Computer Modeling and Simulation (EMS'12)*, pages 395 – 400, Nov. 2012.

Conventions

This section explains how to interpret some plot formats, and defines some specific terms and conventions.

Plot Format

Most of the results in this work are derived from simulations. To reduce the likelihood of showing a simulation outcome which does not represent the most common outcome, we run batches of simulations with some parameters randomised, and with different Random Number Generator (RNG) seeds. We use box plots to show measures of centrality and range of our samples. The following paragraph explains how to read box plots.

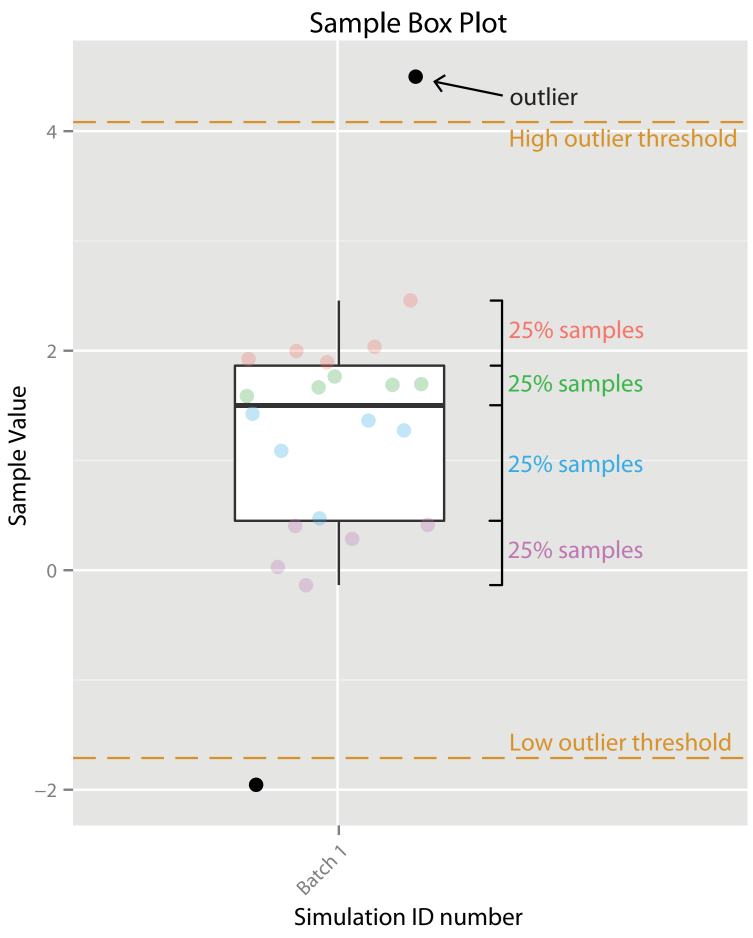
The two "whiskers" (vertical lines) of the box plot span over the lowest and highest quarters of the samples. The lower and upper sections of the "box" cover the two remaining quarters. The dividing line in the box corresponds to the median. Outliers are shown separately, as a dot: a sample x is considered as an outlier if it verifies one of the following conditions:

$$\begin{cases} x \geqslant Q_3 + 1.5 \times IQR & \rightarrow \quad \text{High outlier} \\ x \leqslant Q_1 - 1.5 \times IQR & \rightarrow \quad \text{Low outlier} \end{cases}$$

Where IQR is the Inter-Quartile Range: the difference between the 3rd quartile (Q_3) and the 1st quartile (Q_1), which is also the span of the "box". The following figure shows a sample box plot. We have shown the actual samples in semi-transparent colours for demonstrative purposes: normally, they would not be displayed. The figure summarises 22 samples, and the statistical summary variables are:

```
Min.     1st Quart.   Median   Mean     3rd Quart.   Max.
-2.000   0.4502       1.5030   1.2660   1.8640       4.5000
```

The boxplot shows that the samples with values {-2, +4.5} are respectively considered as low and high outliers. The box plot itself only covers the 20 remaining non-outlier samples: each quartile covers exactly 5 samples.



Sample box plot representing 22 samples. Each individual sample is represented in semitransparent colours (normally omitted).

Storeys in Buildings

North Americans and Europeans use a different numbering scheme when referring to storeys in multi-level buildings. This thesis follows the European scheme: the floor at the ground level is referred to as "*ground floor*", the level above is the *first floor*, etc.

1 Introduction

1.1 Challenges of Emergency Building Evacuations

The hallways and staircases which form part of a building's egress paths are not normally designed to accommodate the surge of evacuees simultaneously leaving the building during emergency evacuations. The congestion which forms as a result increases the building evacuation time, especially if the number of evacuees is high and congestion is poorly managed. The resulting increase in building evacuation time is likely to further expose evacuees to hazards and increase the number of injuries or fatalities. In addition to this, congestion itself can become a threat: in some extreme cases, stampedes or panicked crowd movement can be as hazardous as the event which triggered the evacuation.

Static plan Current building design regulations mandate an evacuation plan which reduces congestion and allows the building to be emptied under a set time (often of a few minutes) when filled to capacity. This plan guides evacuees towards pre-established safety areas by means of static evacuation signs, floor lighting and sometimes fire wardens.
Yet the effectiveness of a static evacuation plan is inherently limited to the scenario it was designed to address. Often, this scenario represents the building filled to capacity: this approach reduces the casualties in the "worst-case" scenario. Yet we argue that buildings are far from being systematically filled to capacity, and if the density of evacuees significantly differs from the one originally assumed, the evacuation plan may be ineffective. During emergency evacuations triggered by bomb threats, explosions or a fire, clearly, every second counts to bring evacuees to safety outside of the building. The building evacuation time should therefore be minimal, not only in the worst-case scenario, but also regardless of the number and initial location of evacuees.

Dynamic system A dynamic evacuee guidance system can overcome the limitations of static systems by responding to each emergency with a tailored plan. Where the number of evacuees is sufficiently high, congestion is bound to occur. Yet we propose to manage congestion in the building by guiding evacuees so that the load on each egress path is balanced. For instance, under certain conditions, it may be judicious to divert a few evacuees through a detour, provided that there is less congestion on that path: the time spent walking through a longer distance can easily be offset if this route is congestion-free and evacuees are able to walk faster. By making optimal use of all paths available, the overall building evacuation time is reduced, as well as the risk of dense crowds forming, both making for a safer evacuation.

However, managing congestion, from a technical point of view, is a difficult task: unlike the length of an emergency path which remains the same at any time, the delay due to congestion on a path is ever-changing, and depends on previous path assignments. The decision-making process associated with diverting evacuees is a complex one: not only must the length of the alternative path be considered, its capacity is also an important factor. For instance, a very short path to an exit will not make a large contribution to the evacuation process unless it is wide enough to accommodate a sizeable portion of the evacuee population. Thus, ideally, the problem requires finding paths which optimally combine short length *and* large capacity; and once these paths are identified, evacuees must be assigned to each path in such proportions that the loads are well balanced.

In this work, we compare two approaches to the problem of routing evacuees in constrained spaces: a "proactive", model-based approach which maintains a congestion forecast, and a "reactive" approach based on real-time congestion measurements. Both approaches are used to estimate the "path delay" metric, which is then fed into the Cognitive Packet Network (CPN), a routing algorithm for Self-Aware Networks, to identify optimal route assignments for evacuees. Our implementation of CPN uses Random Neural Networks (RNNs) to efficiently explore and monitor the building graph, and have exit path recommendations available to evacuees in minimal time. This algorithm can also be distributed to scale to any building size.

Evacuee guidance Guiding evacuees in order to implement complex routing plans is not a trivial task either: they can be directed using exit signs, or guide themselves using personal communication devices. The latter allows each evacuee to be tracked and routed individually, yet displaying the entire route on a screen assumes evacuees are able to read the map and orient themselves as they progress towards the exit. In reality, evacuees may find it difficult and time-consuming to process this information, especially under pressure to leave the building urgently. In contrast, showing smaller pieces of information through displays integrated in the environment (e.g. lighted paths or emergency exit signs), and in a timely manner appears to be a more intuitive and user-friendly solution. We therefore propose to guide evacuees using *dynamic* exit signs whose direction can be controlled, such as the one on Figure 1.1. Yet these signs do not

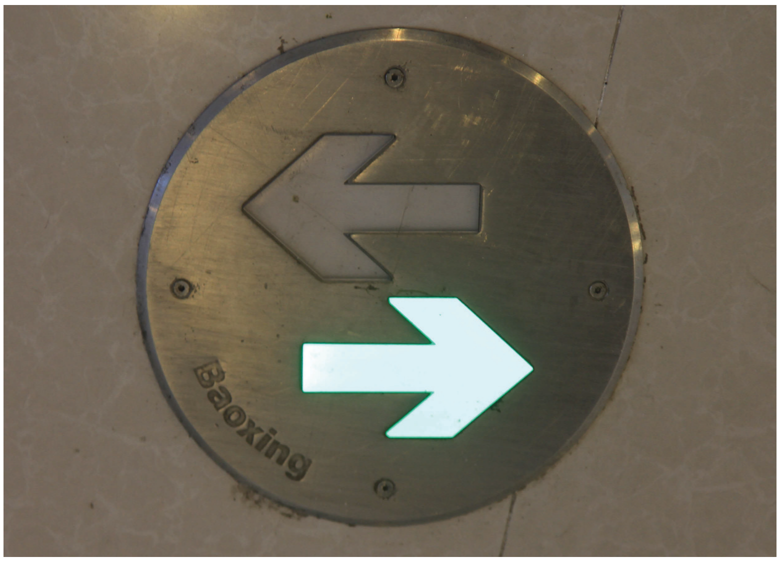


Figure 1.1: Example of floor-mounted exit sign, with multiple pointing directions. Seen in a recently-built shopping mall in China.

discriminate users: all evacuees in the sign's "coverage area" are bound to head towards the same direction, which somehow contradicts our objective to disseminate them across the building. Furthermore, exit signs suppose a "hop-by-hop" routing approach, which is not compatible with some of our source-routing flow-optimisation algorithms. We therefore propose some methods and algorithms to tackle these incompatibilities.

1.2 Summary of Contributions

- We present the first parameter sensitivity analysis of the CPN routing algorithm, using a bench-testing approach. Our analysis explores the initial knowledge gathering phase (convergence), and to a lesser extent the response to changes in the network (latency). By monitoring the quality of the solution throughout the convergence process, we discovered that CPN initially resolves a "backbone" of nodes and edges, and progresses gradually towards "leaf" nodes. We explored the effect of limiting the hop count of exploratory packets. While previous research recommended to set this to a conservatively large of up to three times the network's diameter, we discover that this value can be set much lower. Finally, we show how the parameter which controls the ratio of exploration guided by RNNs over random exploration affects the quality of the solution and the algorithm's latency: randomness brings constant but slow improvement, while the use of neural networks greatly speeds up the discovery process, but is liable to errors due to overtraining.

- We provide a side-by-side comparison of evacuees guidance systems based on a proactive or reactive approach. We find that "raw" performance levels (evacuation times) are comparable, yet we highlight the fundamentally different requirements of each approach. We also find that both approaches have distinct advantages and drawbacks: a reactive approach is adaptive, but requires a constant feed of low-latency sensory readings, and is prone to oscillations – which are difficult to dampen. A proactive approach does not rely on sensory readings during the evacuation, but is sensitive to the evacuee mobility model's accuracy. As it relies on a forecast, it cannot account to any any unforeseen event (at the time the forecast was calculated). Since both approaches are somewhat impractical int heir "pure form", we propose hybrid solutions to overcome their original limitations.

- While the concept of "future capacity reservation" (used for the proactive approach) is not new, we are the first to combine this metric with an advanced routing algorithm (CPN). We find that, unlike approaches using Dijkstra's algorithm, our solution trades optimality for ease of distribution, and better overhead efficiency. Unlike the previous approaches, the solution we propose is also decentralised and distributed: this is a key requirement for emergency assistance systems,a s it means the system can cope with component failures.

- We propose a novel algorithm to control dynamic exit signs based on the output of a routing algorithm which performs source-routing and individually assigns routes to each evacuee. This approach solves the compatibility issues stemming from the fact that dynamic exit signs do not discriminate users and can only display "next-hop" information. Our flow-based approach is based on relatively simple operations, and does not extend the sensory data requirements beyond what is necessary for the routing algorithm. However, it adds several assumptions on the flow of evacuees and the steadiness of routing probabilities which limit its effectiveness.

1.3 Document Outline

The remainder of this thesis is organised as follows. **Chapter 2** provides a literature review of the topics covered in this work: the different approaches to the building evacuation problem and some fundamental theories, solutions for congestion management and load balancing, and evacuee assistance systems. **Chapter 3** is dedicated to the simulation of emergency evacuations. We review some relevant techniques and introduce the platform we will use in this work. We also present and justify our simulation model, and list the aspects of the system we have not modelled. **Chapter 4** covers our evacuee routing algorithm: the Cognitive Packet Network. We detail its mode of operation and present a series of bench-test experiments to fine-tune its parameters for optimal performance. **Chapter 5** presents and reviews two approaches to calculate the “path delay” metric: one uses a congestion forecast, the other one is based on real-time measurements. We compare both metric estimation methods, and feed them to CPN to discover and assign routes to evacuees. In **Chapter 6** we introduce methods to drive the dynamic signs, and analyse their performance through simulation. Finally, **Chapter 7** concludes this work and presents some directions for future work.

2 Background

2.1 Introduction

This chapter is divided into two main parts: the first part aims at providing a basic understanding of building evacuations, its challenges and constraints, and explains how today's building evacuation plans are designed. This leads us into highlighting the limitations of these procedures, and introduce the aims of our research project. The second part of the chapter is a literature review of research in building emergency evacuation, from the fundamental principles of flow optimisation to advanced dynamic evacuee assistance systems.

2.2 Principles of Building Evacuations

Most of today's "best practice" rules in building evacuation are integrated in building codes and regulations, which architects must abide by in order to get their design approved by regulatory bodies. For instance, in the context of sporting venues, the British government's Department for Culture, Media and Sports has been issuing and revising since 1973 the "Events Safety at Sports Grounds" guidance booklet [2], which provides comprehensive safety advice, from the building design stage to crowd management during events. In particular, it specifies an upper-bound on the evacuation time of 8 minutes, regardless of the building's capacity. A similar booklet issued by the FIFA (International Federation of Association Football) requires football stadiums' evacuation time "not to exceed ten minutes" [3].

2.2.1 Fundamental Principles of Evacuation Schemes

In this section, we present a simple and general procedure to validate evacuation plans for buildings, and go on to show its limitations and bias. Our objective is to describe the procedure *succinctly*, therefore we will elaborate on the assumptions and details of the calculations later on in this chapter.
Before calculating the building's evacuation time to ensure it meets regulatory constraints, we first verify that the evacuation scheme produces the lowest possible evacua-

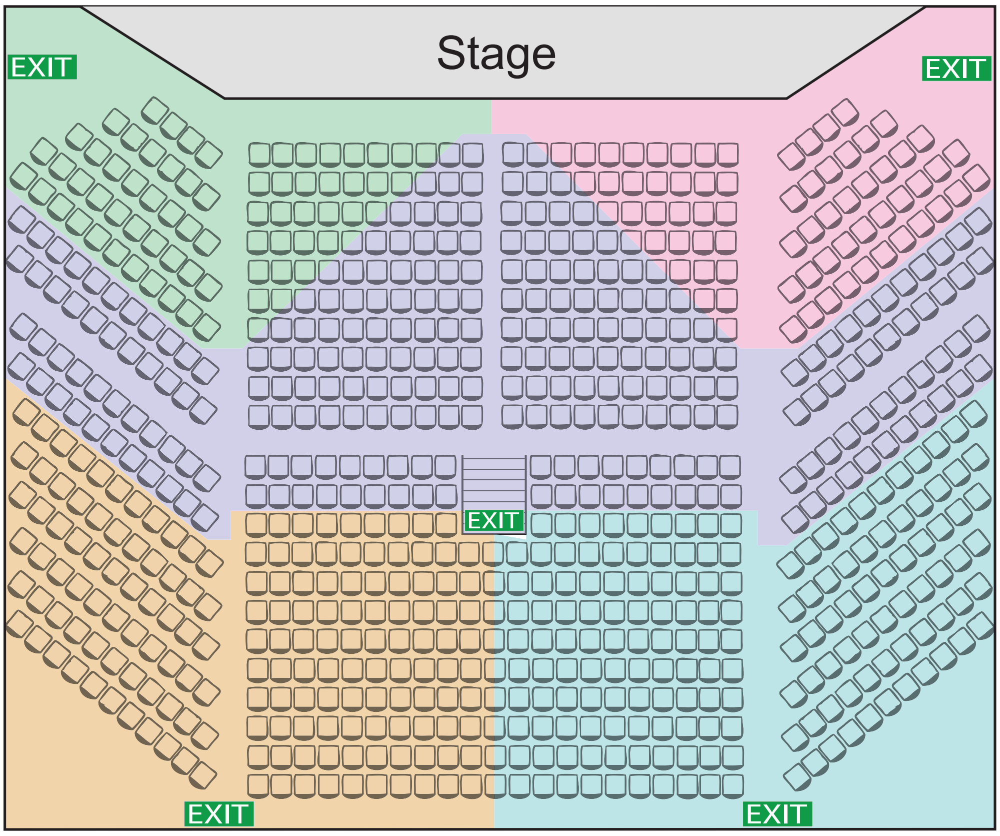


Figure 2.1: Example of Voronoi diagram used to determine exit catchment area.

tion time. We apply Francis' "Uniformity Principle" [36] which states that, to minimise a building's evacuation time, evacuees must be routed so that all evacuation routes are saturated throughout the evacuation, and clear their last evacuee at the same time. This is achieved – under some simplifying assumptions – by assigning a number of evacuees to an exit which is proportional to the exit's maximum allowable flow.

To assess whether the Uniformity Principle is validated, we must estimate the number of evacuees which will use each exit. To do this, we first determine the "catchment area" of each exit by taking into consideration the path evacuees are encouraged to take by the "EXIT" signs in the building. Otherwise, a Voronoi diagram [9] can be used assuming evacuees will head towards the nearest exit. See Figure 2.1, for example.

To determine how many evacuees are in each exit's catchment area, building designers consider a scenario where their building is at full capacity, which makes estimating the number of individuals straightforward: it corresponds to the number of seats or spaces available in each area. The objective is to evacuate the building under a set time: if this can be achieved at full capacity, it will also be for any lesser capacity since the

evacuation time generally decreases with lower attendance.
At this point, the designers are able to evaluate whether the building validates the Uniformity Principle. If this is not the case, the designers may consider adding exits, moving them or increasing their maximal allowable flow to ensure the evacuation is optimal under the Uniformity Principle.
Once the flows are optimised, the second part of the procedure verifies the building can indeed be evacuated within the required time. For this, we estimate how long it will take for all evacuees to walk down their assigned exit path. A simple formula [15] lets us determines the transmission time T of F evacuees through an evacuation path P:

$$T_P(F) = T_P(1) + \frac{(|F| - 1)}{\mu_{min}} \tag{2.1}$$

By applying this formula to each egress path, and for the number of evacuees assigned to each of them, we verify that all exit paths are able to "process" their evacuees within the regulatory time.

The method we have presented is summarised by the flowchart on Figure 2.2. This method has the advantage of being simple, but it also produces limited results. Such a macroscopic-scale model is suitable to provide a quick validation during early-stage design steps, while in the final design stages the designers may consider more sophisticated simulation tools, which we present later on in this document.

2.2.2 Limitations

In the previous section, we have assumed the building was at full capacity to determine the number of people in the catchment area of each exit. This "worst-case" assumption is valid with regards to the objective: if the building can be cleared within the allotted time at full capacity, it can be cleared in even less time if filled to a lesser capacity. Designing an evacuation plan under the full-capacity hypothesis guarantees flows will be optimally distributed in the scenario which carries the highest risk. Yet buildings are not always filled to capacity, and it is unclear if plans elaborated with full-capacity assumed will remain *optimal* when the building is filled *below* capacity. For instance, movie theatre complexes seldom have all rooms full at any time, nor does a university building have all its lecture theatres filled throughout the day, nor are stadiums full at every event, etc. For instance, Figure 2.3 shows the typical attendance at Melbourne's MCG stadium (11$^{\text{th}}$ world rank, by capacity), for an entire sporting season. It is clear that the stadium is only full during the final game of this season, while the average attendance is approximately half of the stadium's maximal capacity.

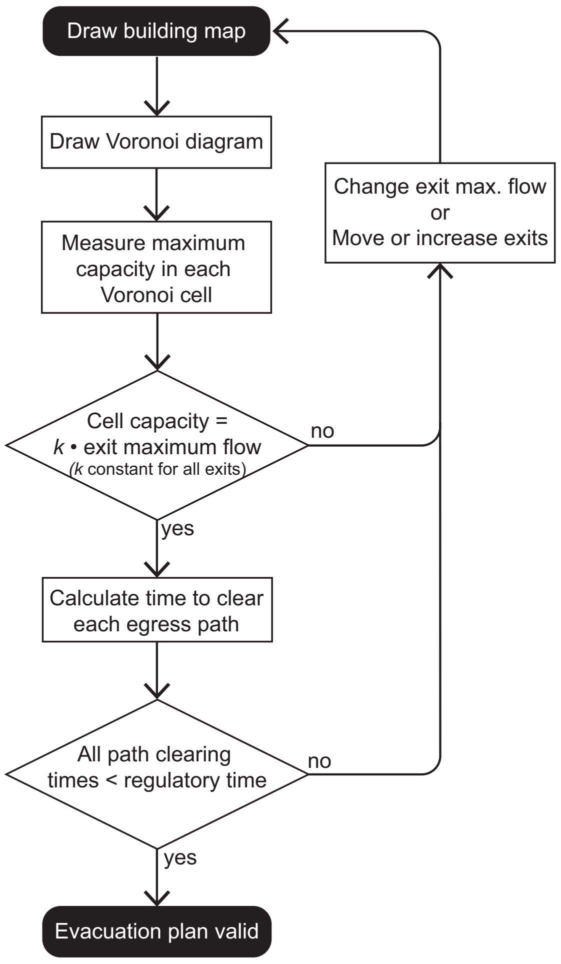

Draw building map
Draw Voronoi diagram
Measure maximum capacity in each Voronoi cell
Cell capacity = k • exit maximum flow (k constant for all exits)
no
yes
Change exit max. flow or Move or increase exits
Calculate time to clear each egress path
All path clearing times < regulatory time
no
yes
Evacuation plan valid

Figure 2.2: Evacuation plan validation process flowchart. The top half verifies that flows are optimised and well distributed on all exits; the bottom half ensures these flows result in evacuation times that are within regulatory requirements.

An evacuation plan made under the full-capacity assumption remains optimal if evacuees are *evenly* distributed across the building. To demonstrate this, let us recall that the evacuation plan is designed so that the number of evacuees using an exit is proportional to this exit's capacity. If the building is not full, but evacuees are distributed evenly, the ratio of occupied vs. non-occupied spaces is constant for any area of the building. Under such circumstances, the number of evacuees using each exit is proportional to the full-capacity number, which is itself proportional to the exit's capacity. However, in a partially filled building, it is reasonable to assume people will not be evenly distributed. In a stadium, they will choose the "best seats", or some parts of the viewing area may be closed to reduce costs. In a movie theatre complex, the rooms where newly-released movies are screened may be more attended. A university building

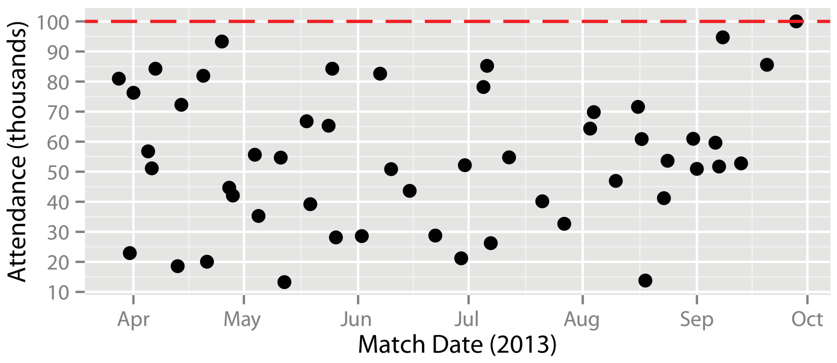


Figure 2.3: Typical attendance figures to sporting events at a large stadium: 2013 AFL season at Melbourne Cricket Ground [5]. The final match is the only time where the stadium is filled to capacity: 100,000, marked by a red line. The average attendance throughout the season is only one half of the stadium's maximum capacity.

may have most lecture theatres filled in the middle of the day, but not during evening lecture hours. All these scenarios create uneven distributions of evacuees across the building. If an evacuation is triggered in these circumstances, inevitably, the exits located near the busier areas will experience much larger levels of congestion than the other ones. The busier exits will take longer to clear, and the building evacuation time will increase as the Uniformity Principle is broken.

In summary, a static evacuation plan is sufficient to evacuate in minimal time a building filled to capacity, or where evacuees are distributed evenly. When this is not the case, the evacuation time will be non-minimal. Thus our research aims at developing a system which is able to minimise evacuation time *regardless* of the evacuee headcount and density. We expect our system to:

- match the performance of a static evacuation plan when the building is either full or has evacuees evenly distributed, or
- outperform static plans in any other scenario, especially where evacuees are grouped near a particular set of exits.

2.2.3 Problem Formulation

Problem As we have seen, static evacuation plans are only optimal when evacuees are evenly distributed throughout the building. When this is not the case, congestion may appear in crowded areas of the building, and lead to non-minimal evacuation times. In the context of emergencies such as bomb threats, explosions or fires, clearly, every second counts in the process of bringing evacuees to safety outside of the building.

An ideal evacuation scheme is one which would guarantee the building is evacuated in minimal time, *regardless* of attendance and evacuee distribution.

Proposed Approach In this thesis we propose to address the shortcomings of static evacuation plans using a *dynamic* evacuee guidance system.

Objective Design an emergency support system which provides guidance to evacuees in order to minimise evacuation time, *regardless* of the count or distribution of evacuees. The system's performance must match or outperform static approaches.

Prior Knowledge We assume that a model of the building is available, and that this model allows an algorithm to compute free-flow transit times, and indicates the maximum allowable capacity and flow on any path. We also assume that we can measure the number of evacuees in all areas of the building.

Constraints The proposed system should, as far as possible:

- Have limited reliance on sensors, especially during the evacuation process where the likelihood of sensor failure increases,
- Be scalable to any size of building,
- Continue operating despite component failures – perhaps in a degraded condition.

2.3 Related Fundamental Principles

As buildings are *enclosed spaces*, the evacuation routes are often *constrained*, that is, their dimensions impose a *capacity*: the maximum number of evacuees which can stand in an area at any given time; and a maximum *flow*: the number of evacuees per unit of time which can transit through a particular section of the building. *Free-flow* occurs when the number of evacuees is sufficiently small so that the maximum flow or capacity is not reached. On the other hand, *congestion* occurs when the number of evacuees exceed the path's capacity: evacuees will have to form a queue, and we describe this path as *saturated*.

Shortly after the evacuation signal is triggered, the movement of evacuees creates a surge of traffic which is generally beyond the capacity of the exit paths. The resulting congestion slows down evacuees and increases the building evacuation time. Therefore, congestion management is a key factor of emergency evacuation optimisation. In this section, we introduce related principles and concepts which apply to capacity-constrained building emergency problems.

2.3.1 Graph Representation

In order to calculate transit times, solve flow optimisation or capacity-constrained routing problems the building must be modelled in a way that algorithms can comprehend. Graphs are a natural candidate for this purpose: their use of nodes and edges abstracts the finer, irrelevant details of the building and reduces its complexity, while retaining important aspects such as transit times and maximum capacity and flow. Several methods exist to convert a building's floor-plan into a graph: the simplest is to divide the building's usable space according to a grid pattern, where a node is assigned to each cell of the grid, and edges connect each adjacent grid cell [108]. This approach has the advantage of being easily automated, which is particularly useful when a high-resolution graph is required, as doing this manually would be a time-consuming exercise. When algorithmic complexity is a limiting factor, it is often desirable to reduce the size of a graph. In this case, a coarse graph is built manually, based on the division of spaces in the building: nodes represent areas which can hold evacuees (offices, classrooms, seating areas, etc.) or points where pathways (corridors, staircases, etc.) intersect. These pathways are represented by graph edges. Such coarse graphs are mainly used when the simulator does not need to model the "interactions" amongst evacuees such as collisions. Chalmet et al. [14] propose a case-study on the decomposition of a classic office building into a coarse graph, and how to estimate the capacity and transit times. The EVACNET evacuation optimisation software's user manual [88] also has a section dedicated to building graph preparation and input into a computer.

2.3.2 Processing Time of a Capacity-Constrained Route

The graph model significantly simplifies the calculation of transit times along a given path. The formula proposed by Chen and Hung [15] allows us to estimate a route evacuation time, that is, the time needed to transfer F units through a path P with a restricted flow:

$$T_P(F) = T_P(1) + \frac{(|F| - 1)}{\mu_{min}} \tag{2.2}$$

where μ_{min} is the edge with the lowest "processing rate". This formula is particularly interesting as it highlights the two components that account for the path's evacuation time: the first, $T_P(1)$ is a constant for a given path and corresponds to the path's "lead-time", or the time it takes for one evacuee to walk along this path without obstruction. The second term is a function of the number of evacuees transiting through the path: it corresponds to the time taken to process all but one individuals through the path's bottleneck. This means that if a path is used by only a few evacuees, the route evacuation time will mostly be influenced by the length of the path, and therefore reducing it will have the greatest impact on the evacuation time. On the other hand, if a path is used by many evacuees, the route evacuation time is dominated by the bottleneck's clearing time. In this case, the evacuation time can only be reduced by increasing this bottleneck's processing rate. This simple formula, however, comes with some strict limitations:

- It assumes the arrival rate is constant, from the first arrival to the last one.
- It assumes processing rates are constant, i.e. users do not slow down depending on the density of the crowd.
- Edges can only process one unit at a time. In cases where multiple evacuees can walk abreast, we create as many separate, parallel sub-paths, onto which we distribute the F individuals evenly.
- This formula supports neither merging nor splitting paths. To handle this, the collection of routes taken by the evacuees must be divided into sub-paths where no merge or split occurs, and be treated separately.

These limitations mean this formula is not suited to precisely estimate the duration of evacuations featuring complex evacuee movements (e.g. leaving a row of seats) in which case formulas based on empirical measures are used [104]. Despite these limitations, complex paths can generally be well approximated: in many cases the processing time through the bottleneck will be predominant over the path's lead-time. For instance, let us consider a multi-storey building where, at each floor, the users departing this floor merge into the flow of users coming down from the staircases. If the number of evacuees is sufficiently large, the evacuation time will be mainly dictated by the processing rate of the lowest flight of stairs.

2.3.3 The Static Maximum Flow Problem

The static maximum flow problem consists of maximising the steady-state flow between a source node and a sink node, in a capacitated network. A theory and algorithm known as "Max-Flow Min-Cut" was found in 1959 by two separate teams: Ford and Fulkerson [35]; and by Elias et al. [26]:

> "The maximum possible flow from left to right through a network is equal to the minimum value among all simple cut-sets" [26].

Where a simple, or minimal cut-set consists of the smallest set of edges that, once removed, disconnects the source from the node[1]. The Ford and Fulkerson algorithm

[1]According to Schrijver [105], this problem was formulated by the military, and their interest was not in the maximum flow but rather in the minimum cut-set: indeed, this set indicates which railway

[35] allows this minimal cut-set to be identified by gradually increasing the flow any available route, under certain rules. This process reaches a stage where the flow can no longer be increased: the edges causing this limitation form part of the minimal cut-set. The Max-Flow Min-Cut theorem and associated algorithm are used to determine where bottlenecks are located in a network, and what its capacity is in steady-state. Beyond this, the fact that this solution applies to a steady-state supposes the process lasts indefinitely. This does fit evacuations well: buildings have a limited amount of evacuees and evacuations are not expected to run indefinitely. In practice, the Max-Flow Min-Cut algorithm's flaw is that it considers *any* path – regardless of its length – yet, clearly some paths should be excluded: for instance, those whose lead-time exceeds the building evacuation time. To solve the building evacuation problem, we must consider path capacity as well as transit times.

2.3.4 Uniformity Principle

Our objective is to design a system which guides evacuees so that the building evacuation time is minimal. The building evacuation time is nothing more than the time at which the *last* evacuee departs the building, therefore, all it takes to reduce it is to make this last evacuee clear the building sooner, possibly by routing it differently. Francis outlines a simple procedure to reduce the building's evacuation time:

> "If the evacuation time for some route j is greater than for all the other routes, then some people using route j could be evacuated by other routes instead, thus reducing the time to evacuate route j while not increasing the evacuation times for the other routes above the time to evacuate route j." [36]

This process can be reiterated until all route evacuation times are *uniform*. If all routes clear simultaneously, shifting an evacuee from one route to another can only increase the building's evacuation time, which indicates that a minima has been reached. This is the foundation of Francis' "Uniformity Principle" of route evacuation times which minimises the overall building evacuation time [36].
The formal definition is as follows: given a set of paths $\Pi = \{P1, P2, ..., Pn\}$, where each path Pn is assigned F_{Pn} evacuees so that all evacuees F are assigned to a path, the uniformity principle is verified if evacuees are routed so that all paths clear at the same time:

$$T_{Pi}(F_{Pi}) = T_{Pj}(F_{Pj}) \quad \forall\ Pi, Pj \in \Pi \tag{2.3}$$

links must be destroyed in order to "interdict an enemy's railway system" with the least number of attacks.

Francis demonstrates that his principle is verified if evacuees are directed to exit paths in numbers which are proportional to the path's maximum allowable flow:

$$|F_{Pi}| = |F| \frac{\mu_{min}(Pi)}{\sum\limits_{\Pi} \mu_{min}(P)} \tag{2.4}$$

Given a set of routes in a building, this principle lets us determine the optimal distribution of evacuees on these paths. However, the assumptions made in order to reach this result limit its applicability:

- This assumes each path is equally accessible to every evacuee, that is, an evacuee has a choice of evacuation paths with comparable lead times. This is unlikely to be the case in very large buildings. Another algorithm is needed to prioritise the assignment of "local" evacuees to nearby paths and reduce the average distance evacuees have to walk.
- We have assumed that paths do not "interact", that is, they neither cross, merge nor split. In order to apply this technique to complex buildings, we must either substantially simplify the model, with a loss of accuracy, or decompose paths into sub-paths, which increases the complexity.

These assumptions limit the effectiveness of this method in complex buildings. In fact, this method is at best suited to provide a quick validation to early-stage designs, or to calculate a lower estimation on the building's evacuation time.

2.3.5 The "Triple Optimisation" Theorem

Jarvis and Ratliff's "Triple Optimisation" theorem expands Francis' principle and compares different approaches used to optimise evacuations. The Triple Optimisation theorem states that:

> "There are three objectives [...]
>
> 1. minimization of the total time, t, to empty the building;
> 2. maximization of the output for the first p periods for each $p < t$;
> 3. minimization of the average time, to evacuate the building.
>
> [...] we are able to simultaneously satisfy all three objectives set out above"[84].

The first objective focusses on the entire building evacuation time. It is the basic principle behind routing solutions which aim to make the last evacuee leave the building earlier – like Francis' approach. The second objective takes a flow-based approach:

by routing evacuees so that the output flow of all exit paths is maximal throughout the evacuation, the building is cleared in minimum time. The third objective takes an evacuee-centric approach: by minimising the average time each individual takes to leave the building, the overall building evacuation time decreases.

2.4 Routing

The general principles we have introduced in the previous section either do not exactly match our problem description, or use a model which is too limited to produce useful results. In this section, we introduce some routing algorithms dedicated to solving the emergency building evacuation problem. The routing component in an evacuee guidance system has three major characteristics:

- the model which represents the routes,
- the way in which the algorithm finds routes,
- the metric: a cost function based on the specific requirements of the application, that the algorithm uses to compare different paths.

2.4.1 Models

The graph approach is a convenient way to abstract microscopic phenomenons and discretise the path of evacuees, while retaining the important factors such as path length, difficulty, transit time, capacity, etc. Yet graphs are "static", and thus unable to represent a system's evolution over time, which limits them to either steady-state or real-time "instantaneous" solutions. For instance, using Dijkstra's shortest path algorithm [22] with fluctuating edge costs means the solutions quickly become outdated, and frequent re-computations are required. Likewise, the Ford and Fulkeson algorithm does not minimise the transit time of evacuees, but only solves the flow maximisation problem in steady-state. Clearly, the problem is not limited to finding high-capacity paths: fast transit times are equally important, and vice-versa. Our problem requires paths which optimally combine high throughput *and* fast transit time.
In order to model this, the static flow graph can be expanded to incorporate the time dimension, which forms a "discrete-time dynamic network flow graph" or more simply, a "time-expanded graph". This model replicates the static graph over a number of discrete time-steps: from the beginning of the evacuation up to a set time horizon T, by which the evacuation should be complete. It is by duplicating the static graph that the model is able to capture dynamic effects, such as the disappearance of some edges after a certain time (e.g. due to fire damage or flood), or the gradual motion of evacuees

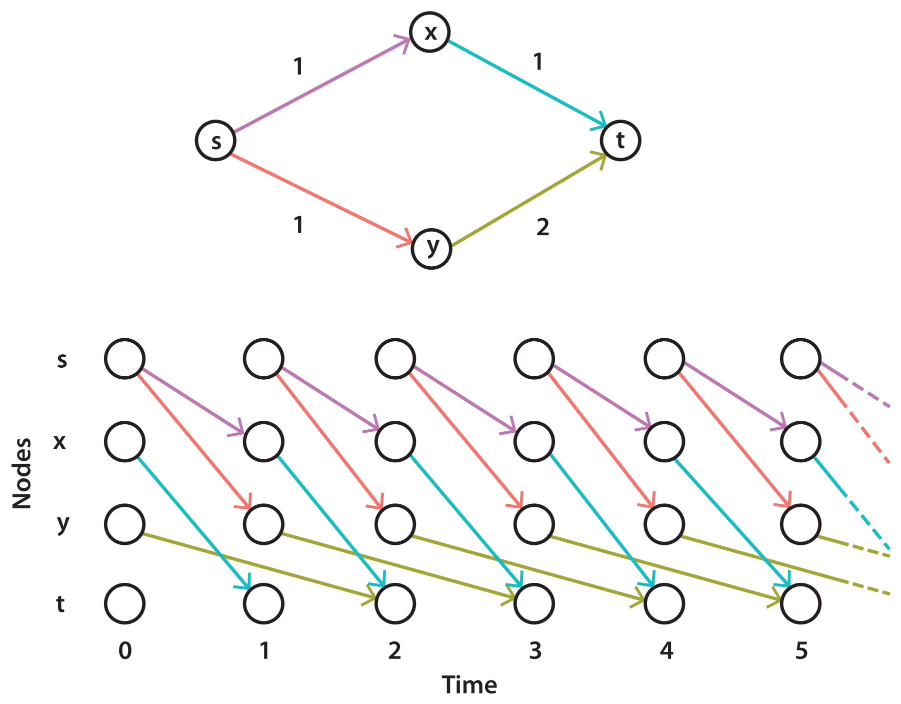


Figure 2.4: Sample time-expanded graph. The static flow graph is above, and the time-expanded graph is below. The time-expanded graph replicates the nodes of the static graph at each step. The nodes are connected by edges which span across different time-steps to represent transit time. For the sake of clarity, we have omitted edge capacities and hold-over edges (which model the possibility of remaining at the current node for some time). Figure inspired from [84].

through the graph. The nodes on each copy of the graph are connected to one another in accordance with the transit delays. Figure 2.4 shows a sample static graph and its time expansion. Another alternative to the time-expanded graph consists of representing the evolution of the graph's features using time-series. In the time-aggregated graph [66], the capacity of nodes or edges at each time-step is stored as time series. Figure 2.5 shows an example of time-aggregated graph.

2.4.2 Path Cost Function

Routing algorithms are often generic: they merely search for paths with the lowest *cost function*. It is the cost function which allows a routing algorithm to compare the fitness of different candidate paths for a specific application. Since our problem is to evacuate a building in minimum time, we could use a cost function which represents

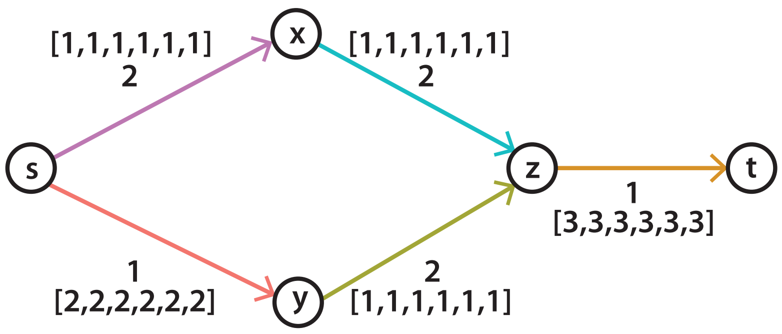


Figure 2.5: Sample time-aggregated graph. The time series associated to each edge represent the available capacity at each time-step, and the edge transit time appears above the time-series.

the time taken by an evacuee to walk along a path: the Triple Optimisation principle states that minimising each evacuee's egress time leads to minimum building evacuation times. However, estimating the evacuation time of a path *ahead of time* is particularly challenging as it depends on congestion. The congestion depends on where the routing algorithm has directed evacuees so far, and perhaps where the next evacuees will be sent. Thus transit time is a *sensitive* metric: it increases with the number of evacuees routed on a path [47]. Since future path delays are undefined, we resort to using some of the following *proxy* metrics.

Distance

Assuming that delays due to congestion are either negligible or affect each path in equal proportion, the average walking speed of evacuees is equivalent throughout the building. At constant speed, reducing the distance walked by evacuees will reduce their egress time. Thus path length can be considered, under certain circumstances, as a *proxy* metric to path transit time. This metric can easily be compounded with other factors to meet multiple objectives, such as the hazard level on a path, to identify the shortest safe paths [68, 29]. However, for evacuees to walk at comparable speeds, congestion must be evenly distributed across the building, which is in contradiction with routing them down the shortest path. This is why shortest-path routing policies often fail as soon as the number of evacuees in the building make congestion inevitable.

Real-Time Transit Time and Congestion

Another approach consists of measuring the current transit time along a candidate path, and considering that this value will remain stable while the evacuee travels along this path. This approach is based on a steady-state assumption: that each routing decision will have negligible effect on the route's transit times. If the path is already saturated, this assumption will be invalid, as each additional evacuee will increase the transit time by joining the queues. In consequence, the algorithm is likely to oscillate under an effect called "self-load" [49]. This oscillation is due to the fact that there is a delayed feedback loop between route assignments and the apparition of congestion along these routes. A solution is to take into account the available capacity of the candidate path, along with its current transit time [42]. For instance, Chen et al. [16] assume sensors are able to count the number of evacuees present in an area and include a measure of congestion to the path metric, using the known corridor width. The resulting metric compounds path length, hazard intensity and a path loading ratio. Since all components of the metric have different units, they are combined using a weighted sum. As expected, the authors highlight the sensitivity of the algorithm to the weights associated to each metric.

Metric Forecasting

The oscillations or frequent route corrections can be avoided by replacing real-time measurements with forecasts. For instance, "safe" paths determined using current hazard conditions may become unsafe as the hazard spreads, which will require route corrections. This can be avoided by increasing the cost of edges near the hazard [109]. However setting an excessively large "safety buffer" zone may block off some vital egress paths too early, while a narrow exclusion zone may be ineffective. A more advanced technique consists of forecasting the spread of the hazard, and if it is likely to reach sections of a path *before* the evacuee has a chance to walk past, then the candidate path is disregarded [12].
In the context of congestion forecasting, Kim et al. have introduced the concept of "future capacity reservation" [66] to forecast the remaining capacity available on each node of the graph at any point in time. Every time a route is assigned to an evacuee, an algorithm calculates the expected time of arrival of this evacuee at each node along the path, and reserves capacity for this evacuee at the expected time of arrival by decreasing the available capacity. The remaining capacity is stored in time series using a time-aggregated graph. For instance, Figure 2.6 shows the remaining capacity of the graph of Fig. 2.5 after routing a flow of two evacuees on the [s,y,z,t] path with an immediate departure. This representation makes it trivial to measure the travel time of an arbitrary path: if there is enough capacity at the expected time of arrival, the flow is allowed on the edge, however if this is not the case, the flow will be held there until such time as capacity becomes available again. For instance, it is clear that routing a third evacuee on [s,x,z,t] would reduce evacuation time: while having a longer travel

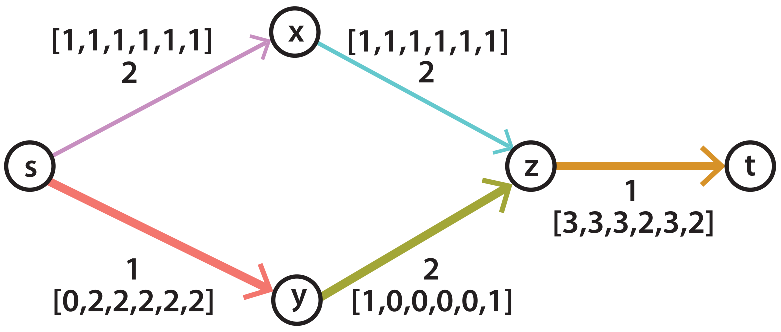


Figure 2.6: Example of the capacity-reservation process after reserving capacity for two units departing immediately on the [s, y, z, t] path.

time, this path has available capacity much earlier.

2.4.3 Route Search and Assignment

Another major distinction in routing algorithms is how they identify egress paths in an unknown graph, maintain this knowledge, and assign routes to evacuees.

Potential-Maintenance

In potential-maintenance routing approaches, an algorithm assigns a potential to each node in the graph. Potentials are set so that the best path is found by performing a "gradient descent" towards the neighbour node with the greatest drop in potential [29, 30]. The convergence process starts with exits advertising the lowest potential (arbitrarily set to zero) and gradually propagates away from the exits. In the setup process, every node evaluates its neighbourhood by adding up the neighbour's potential to the cost of reaching it. The node then compares these values and sets the lowest one as its own potential. Once the process completes, a node's potential represents the cost of the shortest path to the nearest exit. The cost can be set as distance, transit time, or even compounded with hazard level or other metrics. Since each node is able to determine its own potential based on local information supplied by its neighbours, the process is decentralised and distributed. This mode of operation is robust to component failures, which is a desirable feature for evacuation support systems. However, every time the cost of an edge changes (for instance, if congestion appears on this edge) this change in potential has to be propagated along every "upstream" node, which can lead to long update times. In cases where the edge costs vary constantly, the system may be systematically lagging.

Static Graph Approaches

Perhaps the simplest path-searching algorithm is Dijkstra's shortest path algorithm [22]. It can be used as an off-line algorithm with unsensitive metrics such as path length; or as an on-line algorithm using a real-time path delay metric. As the real-time path delay metric is sensitive, it is expected to fluctuate through the evacuation. As a result, prior computations will eventually become obsolete and invalid, and the shortest-path algorithm will have to run periodically to recompute new solutions using the latest route properties. Beyond the limitations of the static graph model presented earlier, Dijkstra's algorithm also conducts inefficient, "naive" graph searches, and it is not suited for decentralisation or distribution. These limitations can be overcome with some heuristic methods [39] or Dynamic Programming [13].

Time-Expanded Graph Approaches

Optimal solutions to *dynamic* network flow problems can be found using the time-expanded graph. For instance, a basic approach consists of expanding the static graph over T steps, and running the Min Cut-Max Flow algorithm [35] from a "super source" to a "super sink": two nodes which respectively connect each source and sink nodes with a set of zero-delay, infinite-capacity edges. The algorithm's output reveals the dynamic path assignments which maximise flow of evacuees *within* the fixed time horizon T. However, this approach suffers from poor scalability and efficiency.

- The complexity of the flow-optimisation algorithms generally depends on the size of the graph: for instance, the Edmonds - Karp algorithm [25] (a slightly more efficient version of the Ford - Fulkeson algorithm) has a complexity of $O(VE^2)$. Yet in order to treat the dynamic problem, we must expand the static graph by creating T copies, with as many time more nodes and edges. This means the problem's complexity no longer solely depends on the graph's features, but also the time horizon.

- The number of time-steps can be reduced by increasing the duration of the basic time unit, yet this reduces the system's temporal resolution, which reduces precision [72].

- The time horizon by which the process should finish cannot be determined ahead of time either. Set too small, there will not be enough time to route all evacuees, and the algorithm will have to run again with a higher time horizon, until the solution is such that all evacuees have been accounted for. This means the number of times the flow-optimisation algorithm will run is somewhat a trial-and-error process with an undetermined number of iterations.

Generally, any linear programming algorithm can also be used to find optimal solutions to any evacuation problem with the time-expanded graph providing a "boundary" for the algorithm's search space. This is the fundamental approach employed by some network flow solvers and evacuation simulators, such as NETFLO [86] and EVACNET+ [89]. An in-depth study of linear optimisation algorithms with application to network flow optimisation can be found in Ahuja et al's book [7]. An overview of the related field of network flow problems can be found in Anderson's literature review paper [8].
These approaches require a cost function tailored to the problem, which lets the linear optimisation algorithm compare the quality of candidate solutions. The evacuation problem can be defined in several ways, for instance, the algorithm we have presented at the beginning of this paragraph solves the following problem: "let as many evacuees out as possible within a set time" (Maximum Dynamic Flow Problem). Other variants of the problem include "minimise the evacuation time, given a number of evacuees" (Quickest Flow) [33], or "find the maximum dynamic flows reaching the exit at every time period $t = 1...T$ (Earliest Arrival Flow) [40, 82]. In their literature review article [72], Hamacher and Tjandra provide an overview of various mathematical problem formulations to the building evacuation problem.
Each problem formulation is associated to a cost function, e.g. average individual evacuation time, building evacuation time, weighted sum of individual evacuation times – where the weights increase in order to penalise later exits. Some functions also aim to meet multiple objectives, such as favouring solutions which avoid hazardous areas, or finding a compromise between reducing evacuation time and minimising the distance evacuees must walk [71]. Linear programming approaches can also be used to compute discrete- or continuous-time solutions [70, 32].
Another approach which produces optimal results without time-expanded graphs is based on the concept of temporally-repeated flows [34]. This concept takes advantage of the fact that a flow is often repeated over several time-steps, from the beginning of the evacuation until the source becomes empty, in order to reduce the complexity of the problem. Hoppe and Tardos build upon this concept and propose an algorithm which computes optimal building evacuation flows in polynomial-time [83], where the complexity does not depend on the time horizon, but rather the number of sources and sinks, evacuees, and parameters of the static graph.
In practice, the complexity of these algorithms is such that they are best suited for relatively small buildings [87]; or to optimise or validate a building evacuation plan during conception stage, where run times of minutes or hours are acceptable. Furthermore, the

imprecisions introduced by our evacuee guidance system and the fact that the speed and position of evacuees is approximative means that an optimal routing solution is somewhat irrelevant. Instead, a solution which trades off some optimality in return for shorter run-times appears better suited.

Time-Aggregated Graph Approaches

The CCRP algorithm (Capacity Constrained Route Planner) [87, 94] uses a modified version of Dijkstra's shortest-path algorithm to search for optimal paths in a time-aggregated graph with future capacity reservations. In particular, the shortest-path algorithm searches for the quickest path, taking into account available capacity at the forecasted time of passage. Reservations are made after each route assignment, and the process of route allocation and capacity reservation is reiterated until all evacuees have a route. As the metric forecasts congestion, the algorithm operates off-line: it computes a solution at the beginning of the evacuation, and this solution remains valid, as long as the expected arrival time calculations are accurate.
The authors determine the complexity of their algorithm as $O(F \cdot V^2 log(V))$ after adding some heuristic simplifications. The complexity of this approach has different parameters, compared to linear optimisation solutions, and cannot be compared analytically. Instead, the authors propose a set of experiments to compare the quality of CCRP's solutions against optimal algorithms (NETFLO [86])and also compare run-times. Their conclusion is that CCRP's output leads to evacuation times which are within 10% of the optimal solutions, with at least a threefold reduction in algorithm run-time.

Self-Aware Routing Algorithms

Self-Aware Networks (SAN) are a class of networks which are designed for decentralised and distributed operation under dynamic network conditions [43, 56]. Instead of performing updates at regular intervals, sensing and measurement are constantly carried out which leads to a steady adaptation to the current conditions. The path search and update is a collaborative effort undertaken by all nodes: dedicated "cognitive" or Smart Packets are forwarded by each node, based on its past experience. Information is also spread over the entire network, so that every node is aware of the best current paths, and previous solutions too. The advantages of SAN are their completely distributed and decentralised operation, and the fact that the required latency can be reached simply by adjusting the rate at which exploratory packets are sent.

Sensible Routing

Sensible routing [47] is not exactly a routing algorithm, but rather a path-allocation scheme. In order to apply Sensible routing, another algorithm must provide several "path options" to route evacuees. Sensible routing consists of assigning more evacuees on the routes perceived as the best and proportionally less on those which are not as optimal. Sensible routing differs from shortest path approaches which generally send *all* evacuees on the shortest path, and create widespread congestion along this one path. Instead, Sensible routing *distributes* traffic over multiple paths, to avoid saturating the best paths and causing their performance level to decrease. As such, Sensible routing is particularly well suited for sensitive metrics where cost increases as paths are assigned more often. Yet this approach has its limitations: determining in which proportion alternative paths should be used is not trivial.

2.5 Evacuee Guidance Systems

In the previous section, we have reviewed routing algorithms, the core "decision-making" component of evacuee guidance systems. Yet this algorithm is only one component of a larger system which also provides computational power, communications, sensory readings, and a media to display guidance and information to evacuees. This section introduces the challenges faced by emergency support systems and reviews some solutions found in the literature.

2.5.1 Cyber-Physical Systems

Emergency support systems can be considered a subset of Cyber-Physical Systems (CPS) [19]. CPS are composed of a network of sensors and actuators which interact with complex, often parallel, physical processes. The challenges which define CPS are the presence of strict timing constraints, and interactions between the different parts of the physical process: both of which also apply to evacuee guidance systems. A recent article [62] relates the challenges faced by CPS to those of emergency support systems:

- Merging information from several heterogeneous sources into simpler, comprehensible decision metrics. For instance, in the context of evacuee support systems, processing the information from several types of sensors (hazard, presence, etc.) and the building graph's information into a simple set of directions for evacuees to follow.
- Communications: informing evacuees, gathering sensor readings, etc.

- Handling partial or fast-changing information: handle the destruction of sensors by the hazard, anticipate the progression of evacuees or fast-spreading hazards, etc.
- The fact that several conflicting action plans exist which meet different objectives, e.g. assisting disabled evacuees or maximising the overall flow of evacuees.

The paper also summarises major approaches to tackle these challenges, such as predictive models, user localisation, use of prior knowledge, etc. Another paper from the same authors [61] gives a very broad overview of the research in the area of human-interacting emergency support systems. This overview highlights that a great part of the literature on emergency support systems is focussed on robustness and fault-tolerance. Reliability, indeed, ranks highly on the specifications list for an emergency support system, since people's lives may depend on it. Achieving low failure rates is particularly challenging in the context of evacuations: the fires, blasts or smoke which trigger the evacuation may also damage or interfere with the components of the system (computing devices, information displays, sensors, transmitters) and damage parts of the building's infrastructure which are critical to the system (data network or electrical cabling, control panels, etc.).

2.5.2 Opportunistic Communications

Opportunistic Communications uses the evacuees and their phones (or communication devices) to form a communication network. This solution has a low reliance on external infrastructure, which makes it extremely robust to component failures. The implementation proposed by Gorbil [68] uses a "store/carry-forward" paradigm to disseminate information on the current conditions in the building. Hazard measurements are made by sensors disseminated in the building, which are transmitted through a short-range link to evacuees passing by. This information is then transmitted from an evacuee to another every time they walk past each other, using the same short-range wireless link. An "epidemic routing" algorithm controls the dissemination process by detecting and dropping stale measurements. A routing algorithm hosted on the evacuee's communication device computes the shortest safe path taking into consideration all hazard measurements collected so far. This path is then displayed to the evacuee on the device's screen. While the low reliance on external infrastructure is an undeniable advantage, since evacuees are used to convey information, their number, concentration and movement will influence the system's performance. The unreliable and delay-prone nature of the network also poses a challenge to the implementation of load balancing or flow-optimising algorithms. Indeed, congestion measurements may change faster than the speed at which evacuees are able to carry the information, and traffic optimisation often requires a comprehensive view of the network, which opportunistic communications do not guarantee.

2.5.3 Gradient-Based Approaches

Systems built around potential-maintenance routing algorithms also benefit from the algorithm's decentralised nature to provide a system resilient to component failures. A typical implementation consists of deploying a network of "decision nodes" which combine some computational power with a hazard sensor and a display unit [30]. As the network does not rely on any "central" node, it will continue to operate as long as the nodes are able to exchange information with their neighbours. Owing to the hop-by-hop nature of the evacuee routing algorithm, the evacuees can be guided by displaying on each node the direction of the neighbour node with the largest potential drop. This guidance method is possibly the most intuitive, as evacuees neither need to read a map nor know their current location, both of which may be difficult to achieve under high levels of stress. This concept is implemented into a prototype system using 20 MICAz motes [109]. However, the rate at which path transit times vary is likely to be faster than the network's convergence rate, and as a result the system may be displaying incoherent guidance most of the time.

2.6 Summary

We have begun this Background Chapter with an overview of how buildings are designed to meet regulatory evacuation and safety constraints. In most cases, evacuees are guided according to a static plan, which meets regulatory constraints in the worst-case scenario, i.e. when the building is filled to capacity. A static approach has inherent limitations: it cannot adapt to *all* possible evacuation scenarios. Therefore we have set ourselves the objective to design a dynamic system able to tailor an evacuation plan to the current conditions and guide evacuees so that the building evacuation time is minimal.
We have then reviewed some capacity-constrained principles and methods which could be incorporated in this system. We have seen that fundamental theorems are often too simple to be directly applied to complex scenarios. While linear programming solutions can provide optimal solutions, they are often prohibitively complex. On the other hand, we found that sub-optimal methods based on congestion forecasts could match our system's requirements.
Beyond the routing component, we have also reviewed some practical evacuee guidance systems and seen that evacuee guidance systems face the same challenges as CPS.

3 Modelling and Simulation of Emergency Evacuations

In the previous chapter, we have provided an overview of evacuee guidance systems and routing algorithms, with a particular focus on flow-optimising algorithms. While routing algorithms are usually validated analytically under simplifying assumptions, the evacuee guidance system itself is often too complex to be validated this way. Furthermore, it is desirable to study the system under complex scenarios, including equipment failure, heterogeneous evacuee motion and behaviour, density-dependant speeds, and much more.
While a real-life evacuation is possibly the most conclusive way to test our system, it is impractical for reasons we will present in this chapter. We will therefore use a simulator to validate our proposed system. This chapter presents the common approaches to building evacuation simulation, and goes on to introduce the DBES: the simulator we will use for this project. We present the relevant details of our model, and also list the parts of our system which are not modelled.

3.1 Validation Tools

While conducting real-life emergency evacuation experiments is possibly the most definitive way to prove the performance of an evacuation support system, this is inconvenient for many reasons:

- It is not entirely repeatable: the behaviour of evacuees changes with each evacuation as they learn and adapt from the previous situations. Their response will be different from one evacuation to another, which limits the ability to compare different solutions.

- It is expensive and disruptive: evacuating an office building or manufacturing plant results in a loss of productivity which may be extremely costly.

- It can be dangerous: even during drills, high evacuee densities can easily result in falls or injuries, especially going down stairs.

Consequently, most prototype evacuation support systems are validated using simulation. The advantages of using simulation are:

- A virtually unlimited number of experiments can be run to test the system's performance under different or identical situations. The parameters can be randomised at each run to determine the range of performance which can be expected from the prototype system.
- The results are comparable: unlike humans whose behaviour may change, simulation allows two different systems to be compared under strictly identical conditions.
- The largest expenses are incurred during the model development phase, from then on, running simulations generally has a marginal cost.

While it is impossible to accurately model human behaviour, simulation is the only viable way to perform repeated or large-scale tests on evacuation systems. In the remainder of this section, we will present some common approaches to modelling and simulating emergency building evacuations. A variety of approaches are used to simulate building evacuations, including flow dynamics, cellular automata, etc., which are covered in detail in Kuligowski and Peacock's review [90, 91] and also in Filippoupolitis' thesis [28]. We will focus this review on simulators which represent the building using a graph, since the evacuee routing algorithm that we will introduce later in this work is also graph-based.

3.1.1 Motion Representation Accuracy

The main distinction between evacuation simulation tools is the level of realism and detail at which they model the flow of evacuees. In the following, we introduce two major approaches: macroscopic and microscopic.

Macroscopic Approaches

Macroscopic approaches are flow-based: they do not model the motion of individual evacuees. This somewhat low level of realism is offset by a lower complexity and quicker computations. These simulators often comprise of a network flow optimiser and solver which searches for the optimal flow allocation and computes the resulting evacuation time. Relevant examples include the EVACNET [89], NETFLO [86] and SAFE-R [69] software.

The major limitation of this approach comes from the simplicity of the motion model, which restricts its ability to produce a realistic output. Indeed, evacuees are assimilated

to “flows”, which inherently prevents modelling heterogeneous individuals with different walking speeds, behaviours, or that start evacuating after an arbitrary time, etc.
With steadily increasing computer processing speeds, the low complexity of macroscopic simulation has become less of an advantage, while their limitations in terms of mobility model realism has led to their decline, in favour of microscopic simulations which can handle much more complex models.

Microscopic Approaches

Microscopic-scale simulators model the behaviour and movement of every individual evacuee in the building. While this method is generally more computationally intensive, it gives complete freedom to assign individualised mobility and behavioural models to each evacuee. The level of realism is usually determined by the resolution of the building graph, and how transit times are calculated. For instance, SIMULEX [108] creates a building graph by overlaying a high-resolution grid (0.25 m pitch) on any accessible area of the building. The physical space occupied by evacuees can be modelled as an ellipse [38] or circles [108], which lets the simulator model the fine “interactions” between evacuees, like attempts to “shoulder” their way through a crowd, etc. These high-resolution simulators are very useful to produce highly realistic simulations of crowd dynamics or understand the behaviour of evacuees going through a bottleneck. In particular, Heilbing has done some extensive research on crowd movements using high-resolution simulators [75]. These simulators are also ideal to accurately determine building evacuation times, and fine-tune the size of exit doors, corridors, etc.
However, this level of precision is computationally expensive, in particular to determine when evacuees will collide. A model which gives a good compromise between complexity and realism is the coarse graph approach with queuing and travel times based on real-life measurements. These travel times can be estimated empirically from measurements made during evacuation drills in the building, or using dedicated formulas [99, 37, 104].

3.1.2 Agent-Driven Simulation

As the evacuee’s motion and behaviour models become more and more intricate, integrating the model into a single-component simulation software becomes extremely challenging from a software development point of view. Agent-based simulation allows developers to represent each simulation entity as a stand-alone “agent”. Each agent’s code scope is then limited to querying to the environment, replying to external queries and acting based on the result of these queries. This allows each simulation entity to be decoupled: not only does this simplify the development process, but since each agent is an independent process, the simulation can easily be distributed over a network of computers and use an agent messaging protocol such as FIPA [4]. Simulex [108], Exodus [41] and DBES [23] are some examples of agent-based building evacuation simulators.

3.1.3 Time Management

Most microscopic-scale simulation platforms use the concept of discrete-event simulation, which is comprehensively covered in Banks' book [11]. This concept requires the simulation process to be broken down into discrete steps, which can be given start and finish "events". For instance, the motion of an evacuee can be broken into discrete steps which correspond to departures and arrivals into edges.
Each entity determines the time of events according to its own model. The simulation engine creates a schedule of all events and advances the simulation time by leaps from one event to another, and notifies the simulation entities when one of their scheduled event's time has been reached. Upon notification, the entities update their status according to their model, and schedule their next event. Discrete-event simulations give the developer freedom to model any element of the model at the required level of precision, while abstracting irrelevant aspects in order to reduce simulation run time. An interesting feature of discrete-event simulators is that they can be made to work in real-time and in conjunction with physical systems. For instance, real sensor nodes are often interfaced with a DES to simulate the movement of evacuees [31, 109].

3.2 Simulator and Model

We have decided to use a dedicated simulation tool: the Distributed Building Evacuation Simulator (DBES) to evaluate the performance of the evacuee guidance systems featured in this work.

3.2.1 DBES

DBES was started in the late 2000s within Imperial College's ISN group, as a distributed, microscopic-scale, agent-based simulation platform dedicated to building evacuations. It has since then been used to validate a variety of evacuee routing algorithms and assistance systems [30] and search-and-rescue algorithms [93]. DBES has also been used in conjunction with a physical sensor network [31], and has run very large simulations distributed over a network of 20 computers with graphs involving over 1000 edges and 720 evacuees [67].
DBES is built upon the JADE (Java Agent DEvelopment) Framework [6], a software library which allows agents to run as independent processes and exchange messages over

a network using the FIPA protocol [4]. This allows DBES to be distributed over a pool of computers and run in a distributed fashion.

3.2.2 Evacuee Behaviour and Mobility Model

In this section, we present the capacity-constrained evacuee mobility model we have implemented in the DBES platform for the purpose of our experiments.

Evacuee Initial Position

The simulator chooses the initial location of each evacuee randomly and uniformly from a list of nodes specified by the building graph. We chose departure nodes which correspond to places where evacuees are most likely to be spending most of their time, for instance classrooms or offices, in the context of a university building.

Departure Time

In real-life evacuations, there is generally a delay between the time when the alarm goes off and the evacuees start to walk towards the exit [106]. For instance, evacuees may put on some warmer clothes and save their work before leaving. We model this by delaying the departure of an evacuee by a random duration, generated from a Gaussian distribution (mean $= 5$, $\sigma = 5$). We use absolute values to deal with negative results: for instance, if the random variable is -5, we apply a delay of 5.

Path Resolution

Our objective is not to determine accurate and realistic building evacuation times: we will mainly use DBES to *compare* building evacuation times under different evacuee guidance systems. Because our analysis is mainly built on comparisons, some errors introduced by model simplifications can be tolerated as long as they equally impact any experiment. In light of this, we consider fine-grained modelling unnecessary, and choose to model the building as a coarse graph where an office, classroom or corridor is broken down into a few nodes. Edges represent the physical paths which allow evacuees to transit from node to node.

Motion Model

We model the motion of evacuees as "entities" travelling through a queuing network. Evacuees are free to go from an edge to another provided that there is capacity available on every edge. If an evacuee wishes to travel along an edge which is currently saturated, it must enter a queue and wait at the joining node until a space becomes available. In summary, the travel time of an evacuee on an edge has two components:

- **Transit delay** The travel time along an edge is fixed and corresponds to nominal values measured in free-flow conditions during real-life evacuation drills.
- **Congestion delay** Since the transit speed on an edge is fixed, we model the decrease in walking speed caused by congestion by forcing evacuees to wait before they can access a saturated edge. In order to access an edge which is at full capacity, evacuees are placed in a First-In, First-Out queue which advances each times a place is freed by an evacuee departing the saturated edge.

In free flow conditions, the congestion delay is null. As congestion increases, some edges become saturated, and the congestion delay serves to reduce the evacuee's average speed proportionally to the levels of congestion. This models congestion, as required for our application, with limited overall complexity and reduced simulation run-time.
The model we have presented does not represent the fact people have different nominal walking speeds. An analysis of real-life evacuations [76] shows the distribution of evacuee speeds narrows as their density increases: indeed, overtaking is difficult or impossible in dense crowds, and thus walking speeds become somewhat uniform. As egress paths mainly operate in saturated regime during emergency evacuations, the assumption of a uniform speed is somewhat valid, at least for able-bodied evacuees, while a separate system would be required for evacuees with special needs. This model could be improved by increasing the evacuee speed's variance in low-density areas.

Behavioural Model

Since DBES is a microscopic-scale simulator, we are free to implement personalised behaviours for evacuees. Our model assumes evacuees are somewhat rational, and they unconditionally follow the guidance system's advice.
While studies show most evacuees *do* tend to follow the directions displayed on exit signs [112], they often simply fail to notice them [113] and instead follow other evacuees [74, 28] or make their own way based on their experience (i.e. the path they took when they came into the building) [106]. Some evacuees may also show some altruistic behaviour (assisting other evacuees) [97], etc. We also assume the evacuation is not competitive. In reality, evacuees are likely to act selfishly [77], this behaviour can be modelled by applying game theory techniques [92]. In the next steps of this research project, we will consider adding realism to the evacuee behavioural model by implementing some of the behaviours listed above.

3.3 Abstracted Components

An evacuee guidance system is a system composed of several components, and this research project does not cover all of them: we have decided to focus on the routing algorithm and the evacuee guidance system. This section lists the components that we have "abstracted", and briefly describes how they could be modelled to produce more realistic simulations.

Localisation component We have assumed there is an *ideal* system which localises evacuees accurately and which is fault-free. Clearly, this is unlikely to exist and therefore our system should be simulated with a more realistic evacuee localisation model. Ideally, this model should incorporate positioning errors which reflect the performance of real-life systems, such as random errors when counting the number of evacuees present at a node, as well as systematic over- or underestimations.
In particular, the localisation system could be modelled based on video techniques [18]. However, the performance of video-based systems is likely to degrade when evacuees form dense and moving crowds (not to mention the effects of smoke during a fire). Another alternative is indoor wireless localisation, which requires evacuees to carry a RFID tag [111] or WiFi [27, 98].

Data network and hardware We modelled a system where data exchanges are instantaneous and no loss occurs. Currently, building safety requirements mandate a dedicated wired network for sensors, while control devices can be networked using open building automation standards [1]. Our model could be improved by taking inspiration from proposed approaches using existing building automation systems [110], or adding a back-up wireless network which could act as a "bridge" when the wired network becomes disconnected due to node failure or damage to the wires by fire, for example.
Our model does not account for processing times either. The algorithms we have presented can be distributed, therefore we considered that computation times can be reduced to a negligible amount, given sufficient distribution. This approach could be improved by constraining the number of devices used to run our system's computations, and modelling realistic computational power levels for the type of hardware considered.

Faults and vulnerabilities This work does not model the effects of exposure to hazards such as fire on the system's infrastructure. This could be improved by modelling damage to power or data wiring, loss of power supply to computational devices, etc. [21, 20]
Our system is also vulnerable to malicious attacks: for instance, from terrorists aiming

to slow down the evacuation of the building they plan to attack. This has been considered in the context of opportunistic communications [68], and we could also design simulations scenarios to include this possibility. In particular, we could model corrupted nodes which constantly advertise their area as congestion-free: this would lead the system in sending most evacuees there. Our systems may also be vulnerable to malicious failures of nodes: the routing algorithm we present in the next section will not route evacuees in the areas managed by disconnected nodes.

3.4 Summary

In this chapter, we have justified the use of simulation to validate our evacuee guidance systems and introduced common approaches to graph-based, discrete-time building emergency simulation. We have then introduced DBES, the simulation tool we will use, and our simulation model. We have justified the level of precision and realism implemented, with regards to the requirements of our application: we believe the model is lean and reduces simulation run-times without making oversimplifications which could impact the validity of our results. In the last section, we have reviewed the aspects of our system which we did not model, and mentioned some possible improvements to provide more realistic simulations.
This research project also gave us a first-hand experience of managing a software development process, and handling large amounts of simulation data. In the appendix (Chp. 8), we reflect on some of the challenges we have faced, and present some case-studies where some off-the-shelf tools were put to contribution to streamline the data management process and reduce development times.

4 Routing Algorithm

The core component of our evacuee guidance system is its routing algorithm: it is in charge of identifying a set of evacuation routes, and determining how many evacuees to assign on each path to obtain the shortest possible building evacuation. We start this Chapter by justifying our choice to use the Cognitive Packet Network (CPN) routing algorithm. We then explain its mode of operation, in particular the concept of Random Neural Networks (RNNs) used to guide the network exploration and update process. Finally, we present a series of experiments aimed at tuning CPN for optimal performance in our application.

4.1 Review of Algorithms

We have introduced and critically analysed a range of capacity-constrained algorithms in the Background Chapter (Chp.2). In particular, we have seen that the linear programming techniques used in conjunction with time-expanded graphs provide an optimal solution, but are not a viable option due to their computational complexity. On the other hand, we see the CCRP algorithm [94] as a valid candidate: it trades some optimality for shorter run times. However, the version of CCRP presented by the authors relies on Djikstra's shortest path algorithm to find the quickest paths in the building. We believe the use of Dijkstra's algorithm in this context is not optimal:

- Dijstra's algorithm is a non-heuristic algorithm and may perform a wide search of the graph at each iteration. This would be appropriate if the graph was expected to change widely and unpredictably in between iterations. However this is not the case: only paths which are assigned to evacuees see a rise in cost. All other areas remain unchanged and searching there for new or updated paths at each iteration is a waste of resources.

- Dijkstra's algorithm is not designed for distributed operation. Its centralised nature means that the server which runs it can be the cause of "single-point failures". Instead, a decentralised and fault-tolerant routing algorithm is preferable.

- In the context of our application, quick computation is crucial to the system's performance. As Dijkstra's algorithm can not be distributed over multiple computers to reduce execution times, this may limit the routing algorithm's ability to scale to larger buildings.

In light of this list of limitations, we have identified the Cognitive Packet Network (CPN), a "self-aware" routing algorithm, as a potential candidate to replace Dijkstra's shortest-path algorithm.

4.2 CPN

The Cognitive Packet Network [48, 63, 57, 55] emerged in 1999 as a routing algorithm concept designed for Self-Aware data packets computer networks. It has since then been applied to several network types: energy-constrained sensor networks [79], integration with IP networks [64], etc.), and problems (access control [59], attack defense mechanisms [52], etc.): a comprehensive overview of the work on CPN can be found in Sakellari's recent literature review paper [101].
In particular, Filippoupolitis [28] was the first to adapt this routing algorithm to provide directions to evacuees leaving a building, by taking advantage of some similarities with the building and packet network models. Indeed, both building and packet network are modelled using graphs, where edges represent links or paths, and nodes are either sources, sinks or intersections where a new direction must be decided upon. The data packets can be likened to evacuees, as they both originate from source nodes and their goal is to reach some sink node (exit). Finally, the overall objective of both algorithms is to determine the optimal path between two nodes with respect to some path cost function, which can be distance, transit time, etc.
The metaphor between evacuees and packets has limits, though: while packets can be guided deterministically, the evacuees have freewill and may decide not to follow the system's guidance. While each packet can be forwarded on an individual basis by routers, evacuees may decide to form a group which cannot be split. Finally, the motion of evacuees is much more complex than that of packets: since evacuees have mass and inertia, they may collide with each other. In particular, contraflow movement (going in a direction opposite to the majority of the traffic) will be significantly slower for an evacuee than it would be for a network data packet, because of physical collisions with oncoming evacuees.

4.2.1 Concept

The CPN concept aims at solving the problems experienced by large networks, where the convergence of an "overall" routing scheme can become a limiting factor. For instance in CCRP, Dijkstra's shortest-path algorithm must be executed in full between each route assignment: while this guarantees a near-optimal shortest path is found each time, this is a blocking and time-consuming process. In contrast, CPN's philosophy is to continuously and probabilistically explore and update the network throughout the route assignment process. Consequently, the routing algorithm never "holds up" the entire process while it updates. Each node takes part in this probabilistic exploration process, by forwarding incoming exploratory packets towards the areas it believes are most worthwhile exploring. This distributed scheme ensures latency is kept low and multiple areas of the network can be updated simultaneously. The amount of overhead sent by CPN can also be tuned to provide an optimal tradeoff between route latency and routing overhead, in accordance with the application's requirements. Another advantage of CPN is that it supports any path cost function, from the shortest and safest path, to compound functions [51], for instance factoring in the "simplicity" of the path, congestion, or weighed more towards safety, at the cost of increased distance, etc. In particular, CPN is compatible with CCRP's path metric based on future capacity reservation.

4.2.2 CPN Operation

CPN relies on a small but constant flow of "Smart Packets" (SP) which are dedicated to network condition monitoring and new route discovery. SPs can be regarded as "virtual evacuees" which explore the network and collect measurements on the route conditions along the way. These SPs are routed on a hop-by-hop basis: each node visited independently decides their next direction. A variety of algorithms can be used to guide SPs, but generally the aim is to focus the stream of SPs in the most worthwhile areas of the network, while limiting random or complete graph searches. To prevent "lost" SPs from wasting resources, they are given an upper limit on the number of nodes they may visit before being terminated: we refer to this as the "maximal hop count". Once a SP reaches its intended destination (i.e. the building's exit), it backtracks along its original path (with loops removed) and shares the information gathered along the way with every node. By gathering information from returning SPs, every node is able to build a "source-routing" route table. These routing tables are constrained in size and ranked by increasing cost, so that the best route has the highest rank in the table. Each time a SP returns with information on a known route (i.e. already in the table), the path cost is updated and the route's ranking is adjusted accordingly. An unknown route is only added to the table if its ranking allows it. The table is also periodically cleaned of "stale" routes whose cost is not regularly updated by SPs: this may happen if the route becomes disconnected, for instance. Using this scheme, every node is able to provide a complete evacuation path (i.e. source-routing) to the evacuees located in its area.

4.2.3 SP Routing

As we have mentioned earlier, the Smart Packets are routed on a hop-by-hop basis independently by each node. A variety of algorithms exist to determine the SP's next hop: the most simple is called "Bang-Bang" [60]. This algorithm assumes SPs have an a priori knowledge of the network topology which allows them to judge if the performance of the routes stored in a node's route table are within the expected range. If they are, the SP follows the direction of the best known route. If the SP "thinks" the routes are not good enough, it decides to explore a new route by selecting its next node at random. This approach is not suitable to our problem as it is impossible to determine a "normal range" of path delays, since they vary widely depending on the number of evacuees in the building.
Another approach is the Sensible routing policy [47], which forwards SPs probabilistically: they are mostly forwarded to neighbour nodes which appear as closest to the exit, and are proportionally less often forwarded toward nodes which are seen as further away. This SP forwarding technique is implemented in Filippoupolitis' work [28]. Approaches based on Genetic Algorithms have also been presented in the literature [44]. Another approach is to run a Random Neural Network (RNN) [45, 46] in each node, and train this neural network so that it can help directing SPs towards the most worthwhile areas of the network.

Random Neural Networks and CPN

Random Neural Networks [45, 46] are a form of neural networks inspired from biophysical neural networks, where, instead of having neuron activity defined as binary or continuous variable, it is defined as a *potential*. RNNs have been applied to various problems, including task assignment [73], video and image compressing [17], and more: a list of application can be found in [100].
An RNN neuron's potential defines the number of positive and negative impulse signals (*spikes*) it may "fire" at other neurons in the network. Firing occurs at random times, according to an exponential distribution. In turn, each neuron which receives a positive or "excitation" spike sees its potential increase, while negative or "inhibition" spikes decrease its potential down to zero. We can probabilistically characterise the type and recipient of a spike emitted by a given neuron: the $p^+(i,j)$ value can be regarded as the chance that a spike emitted from neuron i will turn out to be of the positive type and sent to a neuron j. The $p^-(i,j)$ value corresponds to spikes of negative type. Generally

speaking, the aim is to train the RNN so that the neuron with the highest steady-state probability q_i of being excited (i.e. $k_i > 0$) corresponds to the optimal solution to the problem the RNN is used to solve. This is done by adjusting the $p(i, j)$ values of each neuron, usually throughout a reinforcement learning process.

Guiding Smart Packets Using RNNs

We run RNNs on each CPN node to decide where to forward incoming SPs, much like it is done in data packet network applications [54]. The core concept is to associate one neuron to each outgoing link, and train the neural network so that the neuron with the highest excitation probability correspond to the most worthwhile forwarding direction. Thus each time an SP arrives to a node, it is forwarded to the neighbour node corresponding to the neuron with the highest excitation probability. Once an SP reaches the exit, it backtracks and provides feedback to each node visited along the way, which uses this information to "tune" the RNN through reinforcement learning. Reinforcement learning is done by comparing the performance of the path discovered by the SP to the average past performance: if the RNN's decision appears to have contributed to the discovery of an new shorter path, the neuron associated to the forwarding direction decision is rewarded. On the other hand, if the SP took a longer path than usual, this neuron is punished. In order to prevent neural network overtraining, or getting stuck in a local minima the node may decide to occasionally forward an incoming SP towards a random direction with a set probability [55] (we refer to this as *drift*), in a manner reminiscent of "ant colony" algorithms [24].
The following paragraphs present the major calculation steps of the process of guiding SPs with RNNs. The following is inspired from Gelenbe's original papers on the topic [45, 46].

Initialisation An RNN is created in each node in the network (except for leaf nodes, which include exits). In this RNN, every possible forwarding direction is associated to a neuron. Figure 4.1 illustrates this process: a portion of an arbitrary graph appears on the left, and the right shows the RNN built for node a, where a neuron is associated to each neighbour node. The figure also shows the links between neurons on which excitation and inhibition signals are sent. The rate at which such spikes are sent corresponds to the *spiking weights* w, whose notation is similar to the $p^{+/-}(i, j)$ values, and shown near the link's arrow on Fig. 4.1. The figure confirms that these signals are directional, that is $w^+_{ij} \neq w^+_{ji}$ for example.

We have mentioned earlier that these spiking weights ultimately determine which

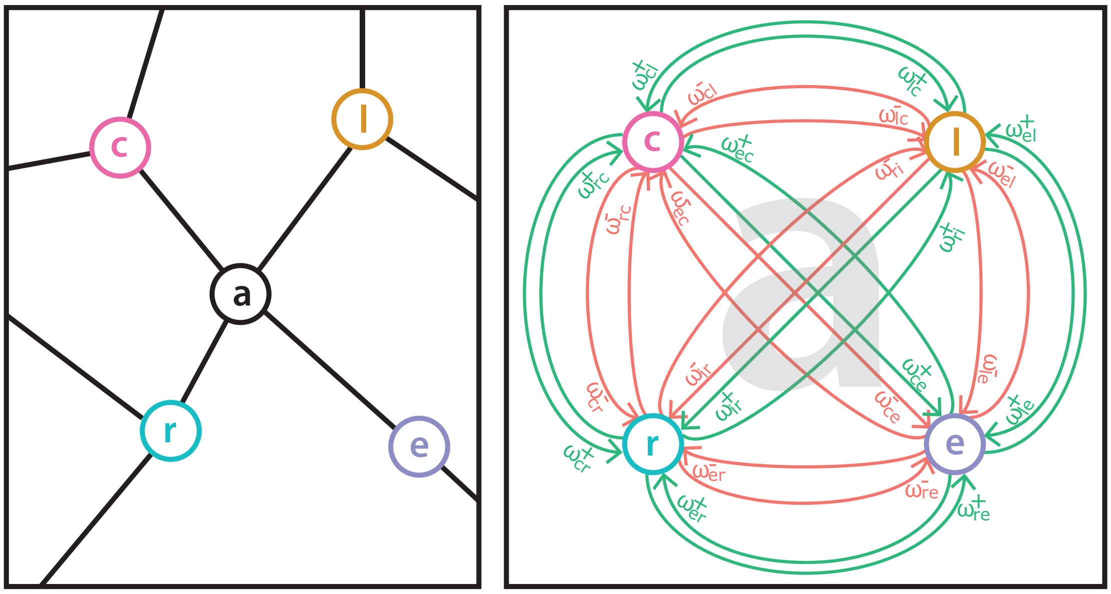


Figure 4.1: The right portion shows a portion of an arbitrary graph, and the right of the figure shows the RNN associated to node a. The figure also shows the spiking weights as long arrows: red is used for inhibitory spiking w^-_{ij}, and green for the excitatory ones w^+_{ij}. For each line, the w notation appears next to the arrow.

neuron has the highest excitation probability and is chosen as the "winner". Since we initialise the neural network with no prior knowledge, by default all neurons must have the same excitation probability, which we aim to initialise at $q_i = 0.5$. To have identical q_i values across the network, we set equal spiking weight values.

For any neuron i in the network, we also define r_i as the sum of all spiking weights coming out of neuron i:

$$r_i = \sum_{m=1}^{n} [w^+(i,m) + w^-(i,m)] \quad m \neq i \tag{4.1}$$

Where n is the number of neurons in the network. We initialise and maintain the value of $r_i = 1$ throughout the RNN's existence. Therefore, to have identical values for all $w^{+/-}_{ij}$ *and* verify $r_i = 1$, the spiking weights are initialised as follows:

$$w^-_{ij} = w^+_{ij} = \frac{1}{2(n-1)} \quad \forall j \neq i \tag{4.2}$$

Reinforcement learning As CPN begins to execute, the algorithm would normally choose the most excited neuron as the "winning neuron" to determine which neighbour node the SP will be forwarded to. However as a result of the initialisation process, any node which has not received any feedback from Smart Packets so far has equal excitation probabilities q_i for each neurons. The algorithm detects that the RNN is unable to provide assistance since there is a tie between all neurons, and randomly designates a winning neuron and forwards the SP accordingly. As a result, SPs initially explore the network randomly while the untrained RNNs are unable to provide any guidance.

As SPs conduct a random walk on the network, eventually one of them reaches an exit, backtracks and informs all nodes visited along the way of the newly-discovered path. The node first removes any loops in this newly-discovered path and discards any part which is beyond this node, i.e. the node only keeps the part of the path between this node and the exit. This path is inserted in the routing table and the corresponding path cost G is calculated, in preparation for the reinforcement learning step. Note that the routing algorithm's objective is to find a path with minimal cost, therefore the less optimal the path, the greater the path cost should be.

The reinforcement learning step begins by determining whether the path returned by the SP is better or worse than previous paths. This is done by comparing this path to the neural network's "threshold" value. The threshold value represents the performance of the paths previously returned by SPs and can be likened to a weighted rolling average of these values. For the l^{th} SP, the threshold value is:

$$T_l = aT_{l-1} + (1 - a)R_l \tag{4.3}$$

Where R is the *reward* associated to the path, which is the inverse of the path's cost: $R = G^{-1}$. We refer to a as the "damping coefficient", where $0 < a < 1$. Viewed from a signals processing perspective, the threshold is a low-pass filter whose cutoff frequency is controlled by a: lowering the value of a rises the cutoff frequency, and the threshold is more liable to "track" closely the variations of the R_l time series. On the other hand, a high value of a dampens and delays the response of the threshold to variations in path cost.

The threshold is initialised to zero: $T_0 = 0$, therefore the path returned by the very first SP is guaranteed to be higher than the threshold. As a result, the neuron j, associated to the neighbour node which was chosen to forward this SP, is rewarded. Note that the excitation spiking weights from j (the "winning neuron") to other neurons are not updated, only the neurons which did not win update their weights. *Excitation* spiking weights towards the winning neuron are increased, and the *inhibitory* spiking weights to all other neurons k are increased. The amount by which the weights are increased is proportional to the improvement made by the new path over the threshold value:

$$\Delta_l = R_l - T_l \tag{4.4}$$

Thus if $T_{l-1} \leq R_l$, if j is the winning neuron and k those which were not selected; every neuron $i \neq j$ updates its spiking rates as follows:

$$\begin{aligned} w^+(i,j) &\leftarrow w^+(i,j) + \Delta_l \\ w^-(i,k) &\leftarrow w^-(i,k) + \tfrac{\Delta_l}{n-2} \end{aligned} \tag{4.5}$$

Throughout the operation of CPN, an SP may return with a path which falls below the threshold, in which case the "winning neuron" is punished, and all other neurons are slightly promoted. Much like the reward process, the winning neuron j never updates its own spiking weights, instead, the neurons k which were not selected increase their *inhibitory* weights to j, and increase their *excitation* weights to all other k:

$$\begin{aligned} w^+(i,k) &\leftarrow w^+(i,j) + \tfrac{\Delta_l}{n-2} \\ w^-(i,j) &\leftarrow w^-(i,k) + \Delta_l \end{aligned} \tag{4.6}$$

The reinforcement learning step is concluded by a normalisation step to maintain the neuron's spiking rates r_i to 1, thereby preventing the w values from increasing indefinitely. Thus each w coefficient is normalised using the ratio between the "new" r_i^* (after the weight update), and the value of r_i before the update:

$$\begin{aligned} w^+(i,j) &\leftarrow w^+(i,j) \cdot \tfrac{r_i}{r_i^*} \\ w^-(i,j) &\leftarrow w^-(i,j) \cdot \tfrac{r_i}{r_i^*} \end{aligned} \tag{4.7}$$

This normalisation step has no effects on the outcome, since it is the size of the w values relative to one another which counts, rather than their actual value.

Determining the most excited neuron Each time a node receives an inbound Smart Packet, it first decides whether to forward it at random, or to use the RNN to guide the packet. We refer to the process of forwarding SPs randomly as "drift" and define the drift parameter b which corresponds to the probability a SP will be forwarded according to the RNN's advice, as opposed to forwarded towards an edge chosen at random:

$$\begin{aligned} p(RNN) &= b \\ p(drift) &= 1 - b \end{aligned} \tag{4.8}$$

If the node decides to let an inbound SP drift, it forwards it to a randomly-chosen neighbour node. On the other hand, to forward the SP according to the RNN's advice, it must determine the neuron with the highest excitation probability q_i. If we consider a neuron as a M/M/1 queue (where k_i corresponds to the queue length), we can leverage some of principles of queuing theory[1] to calculate q_i [58]. The steady-state probability of finding a queue empty is:

$$p(0) = 1 - \frac{\lambda}{\mu} \tag{4.9}$$

where λ is the rate at which arrivals occur, and μ the rate at which departures occur. In our model, a neuron's k_i increases each time an excitatory spike is received, and decreases either when a inhibitory spike is received, or when it emits a spike. A neuron receives excitatory spikes from other neurons at a rate $\lambda^+(i)$:

$$\lambda^+(i) = \sum_j q_j w^+_{ji} + \Lambda_i \tag{4.10}$$

and receives inhibitory spikes from the other neurons at a rate $\lambda^-(i)$:

$$\lambda^-(i) = \sum_j q_j \omega^-_{ji} + \lambda_i \tag{4.11}$$

and emits spikes at a rate r_i. The Λ_i and λ_i terms correspond to the rate at which *external* excitation and inhibition spikes arrive, respectively. In practice, we set these external excitations so that in a "freshly-initialised" neural network, all $q_i = 1/2$.

A neuron's excitation probability corresponds to the probability of *not* finding the queue empty, therefore:

$$q_i = \frac{\lambda^+(i)}{r(i) + \lambda^-(i)} \tag{4.12}$$

Note that determining the excitation of a neuron q_i requires the knowledge of all other neuron's excitation q_j, and all are unknown. In practice, we use an iterative calculation method to solve this problem: we calculate the $\lambda^+(i)$ and $\lambda^-(i)$ values using the initial values of q_i (0.5) or their previous values, if available. We feed these values in the right-hand part of Equation 4.12 for every neuron, and the left-hand gives us new q_i values. These new values of q_i are then again fed into Eq. 4.12, which gives a new value of q_i, and so on. As we iterate this process, the values of q_i converge towards a definitive value [45, 46], and we halt the process when the variation of q_i over an iteration becomes smaller than a set limit.

[1]While RNNs are based on queuing theory, we do not use queuing theory to route evacuees nor to perform load-balancing. The motion of evacuees is modelled as a *queuing system*, but we make no use of queuing theory.

4.2.4 CPN for Evacuee routing Applications

Having presented CPN's mode of operation, let us review its particular advantages for emergency evacuee routing.

Availability

Some of the algorithms we have presented, such as Dijkstra's shortest-path algorithm, gradient-descent algorithms, or linear optimisation produce optimal results, but require a convergence stage, during which no solution is available. This convergence phase corresponds to the time required for the algorithm to explore the network, and to compare different path options until an optimal solution is resolved. As such, the convergence time often increases with the size of the graph. This may be a limiting factor for our application: in the context of emergency evacuations, the routing algorithm must have a solution available within a very short time after the alarm sounds, typically a few seconds at most. Clearly, evacuees cannot be expected to wait in place while the algorithm resolves an optimal path allocation. As the size of the graph increases, maintaining low convergence times will require ever-increasing computational power and fast hardware.

In contrast, the main advantage of CPN lies in its ability to constantly adapt and track changes in the graph, without requiring a process-blocking convergence process. In many cases, CPN will have a solution available instantly – although initially of low quality – and will constantly strive to improve the current solution by sending additional SPs to explore better options and train the RNNs. The quality of the solution provided by CPN will be mainly dictated by the number of SPs sent. This depends on the rate at which SPs can be emitted – which is largely determined by hardware capabilities – and the amount of time CPN is allowed to run before results are required. Thus CPN can theoretically meet any performance requirements, given either sufficient time, or a sufficiently capable network infrastructure.

Decentralisation and Robustness

Another advantage of CPN for our application is its decentralised and distributed operation. Typically, linear optimisation algorithms or Dijkstra's shortest-path algorithm are run from a single server. Not only does this require a large processing power concentrated into a single machine in order to scale with large networks; it also introduces a single point of failure. Thus it would only take the failure of this machine (or its network link) to render the complete evacuee assistance system inoperative. This is clearly an unacceptable risk for our application.

On the other hand, decentralised algorithms are generally tolerant to isolated component failure: the gradient-descent approach, although slow to update, is able to cope with failures, to the point where two "network islands" are formed, or all exit node have failed. The ability of CPN to cope with component failures has been the focus of some research (in particular by Sakellari [100, 103, 102]). While this research used CPN as a data packet routing algorithm, the path discovery and update process is the same for our application and we expect the outcome of component failures will be similar. Their experiments determined that CPN is indeed able to cope with component failure, but operates in degraded conditions. When a CPN node fails, it ceases to forward SPs. After a short moment, the RNNs in the neighbouring nodes realise that all SPs sent towards the failed node never return, and punish the neuron associated to the failed node to prevent more SPs from being sent there. As a result, CPN stops exploring the area covered by this failed node and automatically shifts its focus on other areas of the network instead. This, however, means that any route which passes through the coverage area of a failed node will no longer be considered. If this node connects two sub-networks, its failure will create two disconnected "network islands". Likewise, the failure of nodes covering the building's exits would severely impact our evacuee guidance system's performance, as CPN would not route any evacuee through these particular exits.
In summary, CPN is able to cope with node failure, but to guarantee an acceptable level of performance we recommend hardening or adding redundancy at nodes which cover areas that "bridge" different areas of the building (staircases, footbridges, etc), as well as any node covering a building exit.

4.3 CPN Parameter Optimisation

The previous section revealed several parameters which have the potential to affect the performance of CPN, in particular:

- Smart Packet drift parameter: the probability that a SP will follow the RNN's advice over a random next hop,
- Smart Packet's hop limit: the number of nodes a SP is allowed to visit before being considered as lost or unsuccessful and terminated,
- The damping coefficient which determines how a new route cost affects the RNN's threshold.

The aim of this section is to determine how each parameter influences the routing algorithm and ultimately tune them for optimal performance. We are also interested

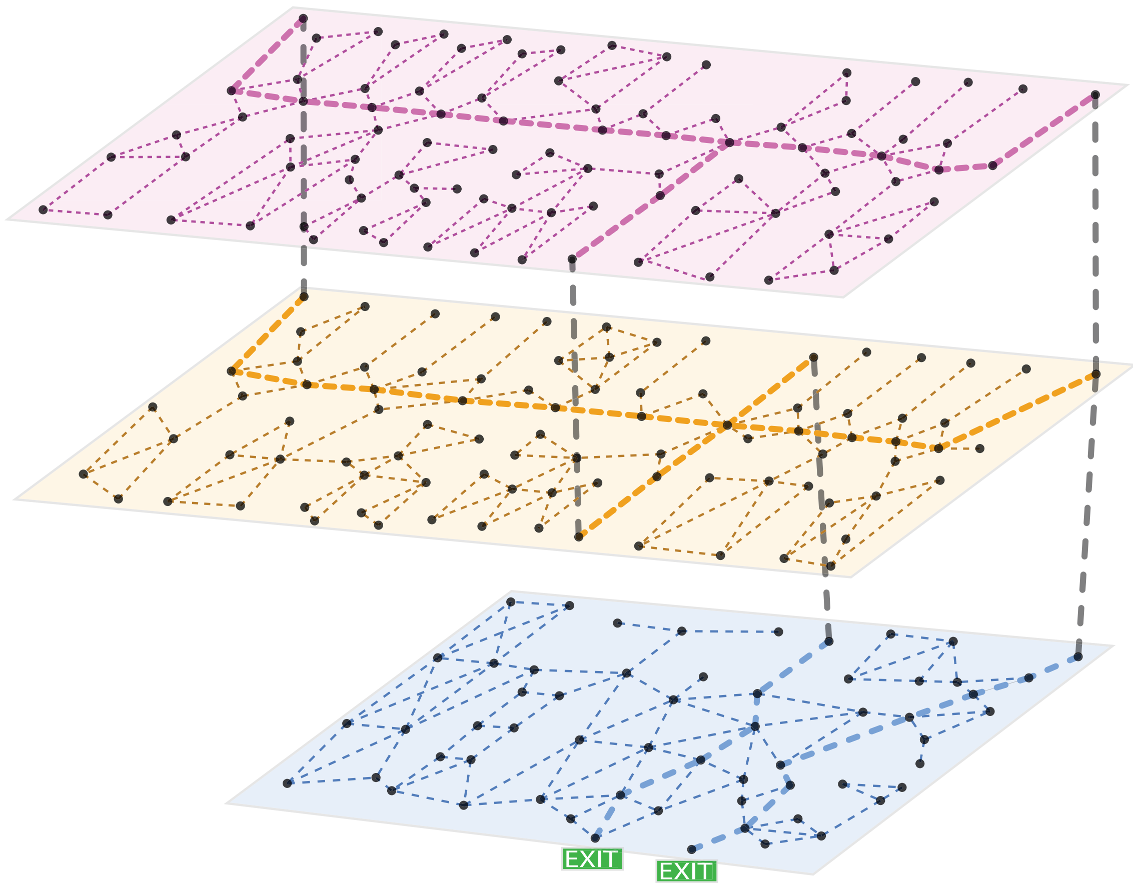


Figure 4.2: Graph representation of the building model. The two signs on the ground floor mark the position of the building's exits. The bolder edges show the major evacuation paths.

in adjusting these parameters to reduce CPN's latency when route costs change. We will do this by bench-testing CPN with different parameter values. For the sake of simplicity, we use path distance as routing metric. The tests will be run on the same building graph that we will use in the remainder of this work. This building graph consists of 240 nodes and approximately 400 edges and represents the three lower floors of Imperial College London's EEE building. The building combines office space and classrooms on the upper floors with a large lobby area on the ground floor, where the two exits are located. Figure 4.2 gives a 3-D representation of this graph. A "functional" building map, showing offices and hallways can be found in the Appendix, see Figure 8.3. For reference, each floor of the building upon which this graph is based has an area of approximately 1000 m^2.

4.3.1 Initial Route Resolution

This section is dedicated to the algorithm's initialisation phase and initial knowledge gathering. Our aim is to determine how fast CPN is able to identify optimal routes, and to identify which parameters influence this process. The initial convergence process is particularly important to our application: the evacuee guidance system can only start advising evacuees on optimal egress paths once the routing algorithm has identified them.

In this experiment CPN is initialised without any prior knowledge of the network, and its objective is to find the shortest path to the nearest exit from *every* node in the graph. CPN starts by allowing every node to send SPs without any particular order: our algorithm chooses a node at random, sends a SP from that node and reiterates this operation indefinitely. For convenience, we define a "batch" of SPs as the transmission of 240 SPs which corresponds to the number of nodes in the graph[2]. The experiment is terminated after 39 SP batches. Between each batch, we take measurements from every departure point: we collect the best route found by CPN so far. Each of these routes are compared to the corresponding shortest path computed using Dijkstra's algorithm. The metric expressing the route quality is the ratio Q_P:

$$Q_P = \begin{cases} 0 & \rightarrow \quad \text{if CPN has not found a route yet} \\ \frac{length(shortest\ path)}{length(CPN\ path)} & \rightarrow \quad \text{otherwise} \end{cases}$$

Unless the drift parameter is set to 1, the SP's path will have an element of randomness: the "drift". Nodes also send SPs at random intervals. We therefore run 10 iterations of each experiment to obtain a broad representation of the *normal* process behaviour.

Drift Parameter

We start by analysing the sensitivity of the drift parameter variable. To illustrate the expected effect of the drift parameter on CPN's performance, we propose two trivial case-studies using the parameter's extreme values:

- **Drift Parameter = 0** In this configuration, the SPs choose their next hop at random and effectively perform a random walk of the network. The probability $p(P_N)$ of going through the path P_N formed of the collection of nodes N is:

$$p(P_N) = \prod_{n=0}^{N-1} \frac{1}{d_n} \tag{4.13}$$

[2]Since the algorithm chooses which node will send an SP randomly, after one SP batch some nodes may have been chosen more than once, while some may not have been chosen at all.

Where d_n is the *degree* of the node n, i.e. the number of connections to other edges. This equation indicates that any finite path has a non-zero probability of being visited by an SP performing a random walk – as long as the path does not exceed the SP's hop limit. We can see that $p(P_N)$ decreases with N and d_n, that is, the path is less likely to be visited if it involves more nodes and if these nodes have a high degree.

- **Drift Parameter = 1** As we have mentioned earlier, a newly-initialised RNN which has not received any feedback has all its neurons in a "tie" with $P_i = 0.5 \forall i$. This tie is broken by choosing one of the contending neurons at random, thus SPs explore the network at random while RNNs are untrained.
  As soon as a SP discovers a valid path, all neurons along that path receive positive reinforcement and become the most excited neurons. Because subsequent SPs are not allowed to drift, they will follow the most excited neuron, which corresponds to the path of the first SP, and further reinforce it indefinitely. Effectively, CPN will only resolve one path per departure node, and there are no guarantees on its optimality, since it was discovered through a random walk and is never further optimised through random exploration.

From these two case studies, we can infer that a low drift parameter guarantees that the optimal path will be found, however the process might be extremely slow since the knowledge gathered by the RNN is disregarded. On the other hand, higher drift parameter values ensure a solution will be reached rapidly as information gathered by previous SPs is re-used to guide the next ones. However, this initial solution may be sub-optimal, and further improvement may be slow or limited because of an over-training phenomenon: the algorithm does not promote enough random exploration, instead, the same original sub-optimal path is reinforced over and over.

Route quality Figure 4.3 illustrates the path resolution process for three representative values of drift parameter. Note that the preface of this thesis contains an explanation of how to interpret "box-plot" graphs (*Conventions* section). Initial data analysis showed that most of the improvement in path length is made during the early stages, and the solution reaches a plateau after the first few SP batches. We therefore use a pseudo-exponential scale on the horizontal axis.
The first graph illustrates the "slow-but-steady" learning process associated with low drift parameter values: over a quarter of the departure nodes are still left without any route by the second SP batch, as most of the SPs get "lost" and reach their hop-count

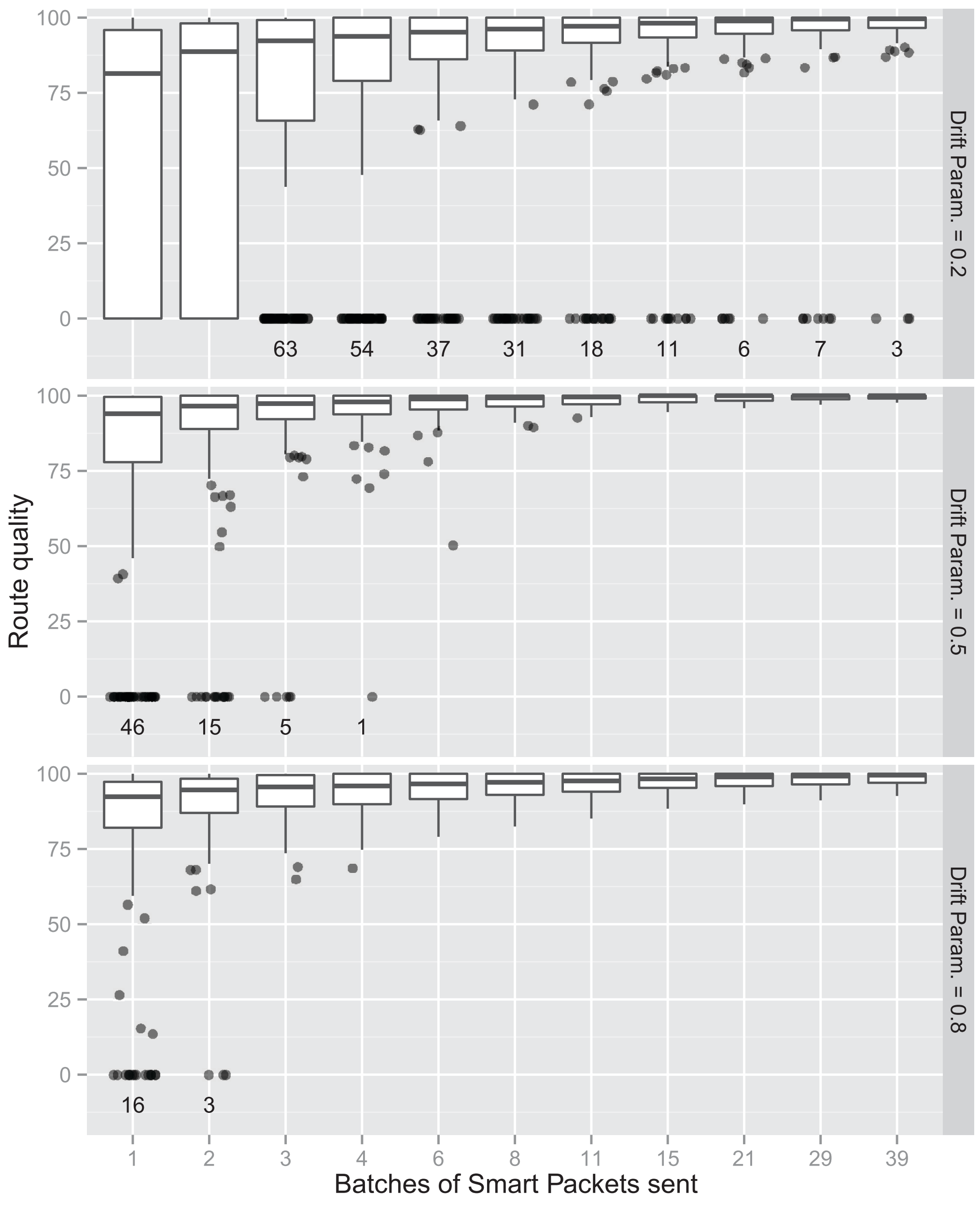


Figure 4.3: Quality of routes found by CPN, from all departure nodes. 100% corresponds to the shortest path, lower percentages indicate proportionally longer routes. A score of zero is attributed when no route is resolved yet (the count of unresolved routes is shown below each boxplot). One SP batch corresponds to 240 SPs being sent from randomly-chosen nodes. The top graph shows the “slow but steady” learning associated to low drift parameters, while the bottom graph displays the “quick but approximate” discovery associated with high drift parameter.

limit. Some nodes still have no path resolved at the end of the experiment, yet it is clear that CPN continues making small but continuous improvements at each step as the box plot gradually shrinks around the 100% path performance mark. On the other hand, the bottom graph on the Figure (high drift parameter) exhibits the quickest initial path resolution process: CPN has resolved a path for all but two of the 240 departure nodes after sending only one batch of SP. However, compared with the middle graph (drift parameter = 0.5) the median takes longer to reach 100% in the long-term, and the box plots remain wider at the end of the experiment, confirming that the algorithm stagnates with sub-optimal values in the long-term. Finally, the graph in the middle of Figure 4.3 shows a "middle ground" where some of the initial resolution speed is "traded off" for a sustained higher improvement rate: while it takes 5 batches of SP to resolve paths from every departure node, the quality of the routes, by the 8th SP batch, is highest across all three experiments presented.
Our intended application – finding evacuation paths – requires a fast path discovery: it is unacceptable for evacuees to have to wait at their departure location for the routing algorithm to resolve a path. It may be preferable to compromise with a slightly sub-optimal path, if it is available sooner. This encourages us to set a high drift parameter value. However this is not a definitive conclusion: as we will see, the drift parameter also influences the algorithm's latency in dynamic conditions.

Spatial convergence Having considered the overall convergence process of CPN, we now analyse how the convergence occurs from a spatial point of view. Our aim is to see if there are discrepancies in convergence rate based on a node's location. The figures arranged in Table 4.1 show a total of 9 representations of the building network. Vertically, the building map is represented at three different stages of the resolution process: after sending the 1st, 5th and 10th batches of SP. Horizontally, we can compare the results for three different values of drift parameters: {0.1, 0.5, 0.9}. Each cell contains a "flattened" view of the building graph, with the highest floor on top, and the ground floor at the bottom. The graph is colorised so that black is associated to $Q_P = 0\%$ (no path), white is associated to $Q_P = 100\%$ (optimal path) and shades of grey correspond to intermediate values.
The figures corroborate the previous findings: a low drift parameter leads to a very slow route discovery process, yet we see that significant improvements are made after each batch of SP. A high drift parameter leads to a faster initial discovery; but by looking at the last row (after 10 SP batches) the graph corresponding to the highest drift parameter (0.9) is slightly darker than the one in the middle (0.5). This confirms that in the long term, lower drift parameters achieve better results. Looking into further detail, we see

that CPN converges at a faster rate in the following areas:

- **Around exits** The likelihood of identifying an exit path during a random search is increased for shorter paths. Therefore, SPs starting near the exits are at an advantage, as they are closer to the target. This explains why route quality generally resembles a gradient function, where the quality reduces as we move away from the exits.

- **Main egress routes** The RNNs focus SPs towards what they identify as the most promising areas, therefore main egress routes are most frequently visited. We recall that the information gathered by a SP is not only available to the node which issued it, but is also shared with every node visited along the way. Therefore, while leaf nodes can only rely on "their own" SPs to gather information on the network, nodes located near the main exit paths will harvest information from the many SPs passing by. For instance, we can see that the main corridors (horizontal edges in the middle of the first and second floor) are lighter than some other areas which may be located closer to the exits. This is particularly visible on the top row, centre and right cells.

The spatial analysis of the convergence shows a desirable feature of CPN for our application: CPN rapidly establishes a "backbone" of high-quality main exit paths, and gradually explore more intricate areas. This ensures that evacuees, regardless of their location, will not have to walk too far to find a node with a high-quality path able to guide them to the exit.

Hop Count Limit

Having analysed the impact of the drift parameter with very high Hop-Count limits, we now reduce this parameter and observe the impact on the path quality. There is little guidance in the literature on the optimal value of this parameter, apart from setting a conservatively large value: two or three times the diameter of the network [54]. We have determined that the graph contains no nodes located further than 23 hops away from an exit. Consequently, the CPN algorithm cannot fully converge with SP hop count limit values lower than 23. We run a series of experiments similar to the ones presented earlier and use the same Q_P metric for path quality. Figure 4.4 shows the result of this experiment in the same format as Figure 4.3[3], where the SP's maximal hop limit is the

[3]While the middle graph on figures 4.4 and 4.3 is generated from the same dataset, the outliers may be at different positions, since we take a random sample of outliers to represent the typical outcome of a *single* run, out of the ten runs in the experiment.

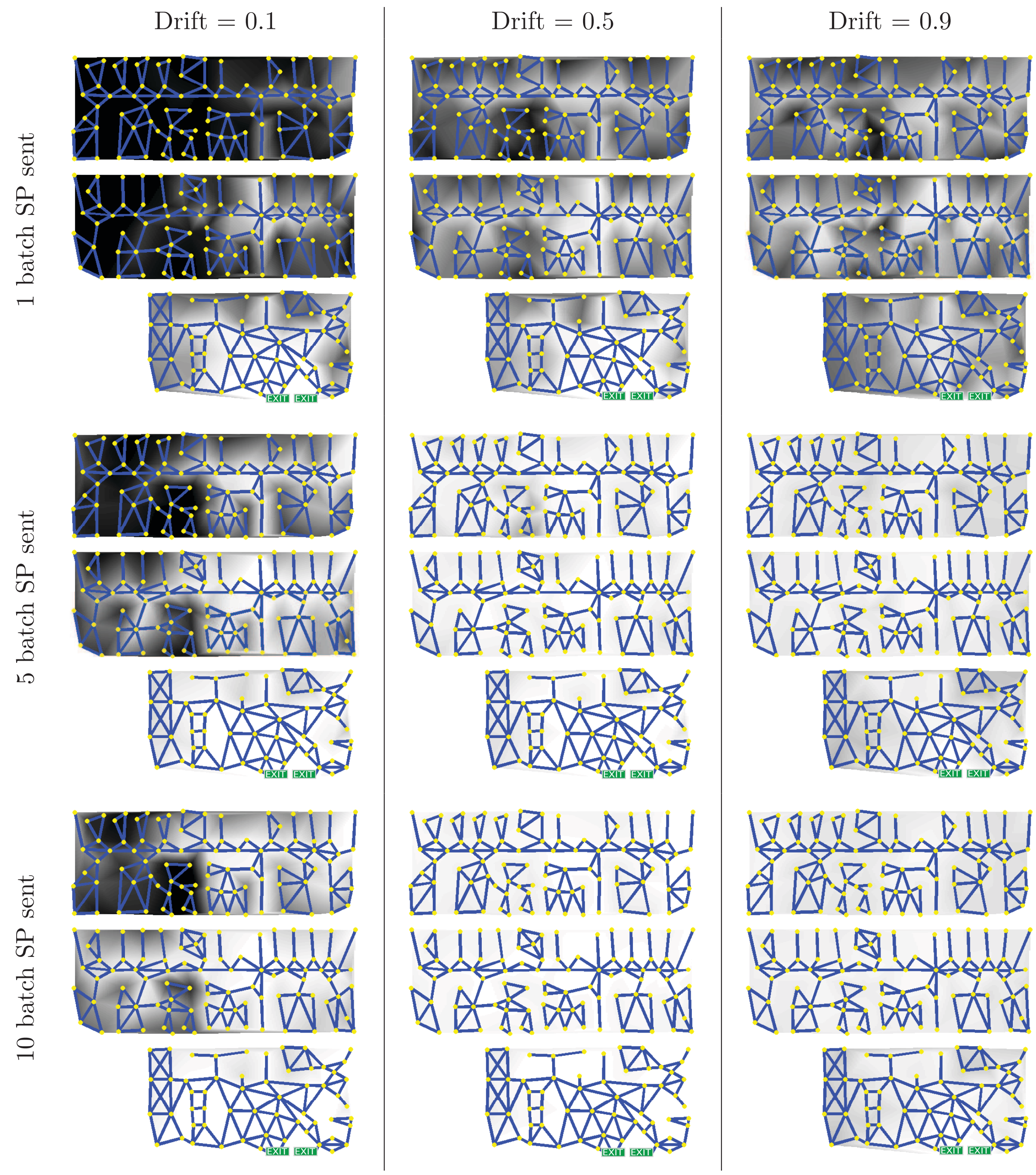


Table 4.1: Initial path resolution process across the three floors of the building (ground floor at the bottom). The lighter the colour the better the path found by CPN. Black areas correspond to departure points where CPN has not resolved a path yet. The figure shows CPN starts by resolving a "backbone" of high-quality paths (main egress routes), and gradually progresses towards leaf nodes. The figure also confirms previous findings: low drift parameters leads to slow but steady discovery, while high drift parameter resolves paths quickly but stagnates with sub-optimal solutions.

experimental parameter with values in {30, 45, 60} and a set drift parameter value of 0.5.

The major information derived from Figure 4.4 is that imposing a short limit on SP hop count makes it harder for CPN to find routes. Indeed, SPs are less likely to randomly find an exit if the hop-count limit is low, since any detour or loop will result in their premature termination. As no SP manages to reach an exit, the RNNs are not provided with any feedback and remain useless while the random exploration continues. However, past a very slow beginning, the long-term optimisation process seems relatively unaffected: eventually some SPs will reach an exit and provide feedback to some RNNs, which "activates" the guidance process for the subsequent SPs and increases their success rate, and the route optimisation and discovery process becomes less affected by the hop count limit.

Yet there is also a drawback in increasing the SP's hop count limit: computational overhead. Allowing SPs which are effectively lost to continue their exploration means that – when they eventually reach the exit – every node they have wandered to will have to process the information gathered by the SP, which is often of little value. This includes a reinforcement learning process on each RNN, which, computationally, is the most complex part of CPN [46]. Figure 4.4 also suggests that the benefits of increasing the SP's hop limit progressively fade: while the convergence speed improves greatly by increasing the SP hop limit from 30 to 45, increasing it from 45 to 60 has comparatively marginal effects.

In conclusion, the hop limit should be set well above the length of the longest path to resolve, while keeping in mind that *1)* this increases the computational cost and *2)* the improvements to the convergence process tend to fade as the SP hop count is further increased. Our analysis is based on experimental results of the algorithm on a particular graph. We cannot generalise these results beyond the guidelines previously mentioned, as we estimate the optimal SP hop limit is likely to depend on particular features of the graph, in particular the average node connectivity.

4.3.2 Reaction to Updates

With CPN's initial route discovery process characterised, we now move to dynamic operation, in particular how sudden changes in route costs are handled. CPN will be used as a flow-optimising routing algorithm, and is expected to closely monitor congestion on several evacuation paths and make frequent route changes in order to distribute the load of evacuees optimally over each egress path. The experiment presented in this section is designed to measure how long it takes for CPN to switch to an alternative route when

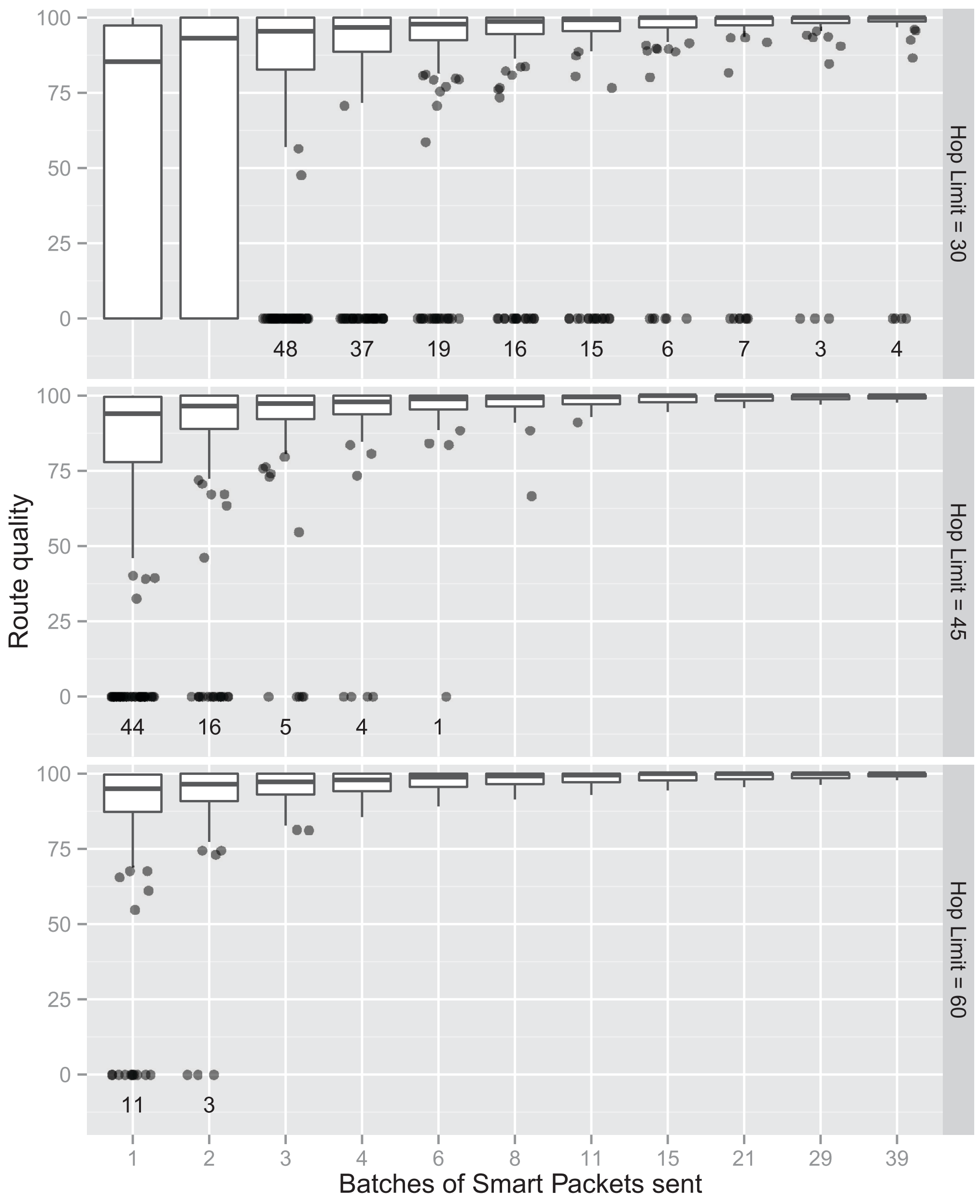


Figure 4.4: Convergence process of the CPN algorithm based on the number of SPs sent. Figures shown for different values of the SP's hop limit. The drift parameter is set to 0.5. The figure shows how low hop limits initially slow down the learning process, however, long-term progress is relatively unaffected thanks to the RNNs' guidance.

the cost of the original route increases, and to identify the parameters which control this process. A similar analysis conducted by Gellman and Liu [65] determined that increasing the rate at which SPs are sent reduced the latency, however the influence of other parameters is not considered.
We propose a two-phase bench-test experiment to analyse the routing algorithm's performance in dynamic conditions. We first allow CPN to fully converge by sending 20 batches of SPs. In a second phase, we artificially increase the length of the thick red edge on Figure 4.5. This edge represents the eastern staircase leading to the first floor, and forms part of the evacuation path (red path) from the nodes highlighted in purple. We have chosen this edge as bypassing it is not trivial, and the edge's increase in length is such that the green path will become the new shortest path. Thus the goal of our experiment is to observe how long it takes for CPN to recognise the green path as being the best, from the purple nodes: when this happens, we record the number of SPs sent so far. The number of SPs sent before the route change occurs gives us an insight into the update latency of the routing algorithm in the presence of dynamic path metrics. The experiment is aborted if CPN has not updated after sending 2000 SPs (8.3 SP batches). We found that the update process is faster than the initial convergence process, and thus to obtain a suitable resolution we increase the measurement frequency to once every 10 SPs.

The data we accumulated during this experiment suffices to meet our objective of measuring the algorithm's latency, however it is often not detailed enough to reveal the intricate details of the route switchover process. Indeed, route update is a complex probabilistic process, due to the partly random motion of SPs (drift), the fact that each node forwards SPs independently and the complex RNN interactions. When possible, we formulate some hypotheses to explain how a parameter affects the dynamic behaviour of CPN, but to validate these hypotheses, we recommend conducting a statistical analysis of the path taken by SPs and monitor the evolution of the excitation probabilities during the route update process. Such an in-depth study of the CPN process is beyond the scope of this work, therefore we will limit ourselves to an analysis of the experimental results and determine optimal parameter values empirically.

Drift Parameter

We first analyse the update process based on the SP's drift parameter. Figure 4.6 shows the empirical Cumulative Distribution Function (CDF) of the route update process, that is, the probably $p(n)$ of CPN having switched to the green path by the n^{th} SP. The figure shows the average of 20 experiments for each drift parameter value, and the SP's

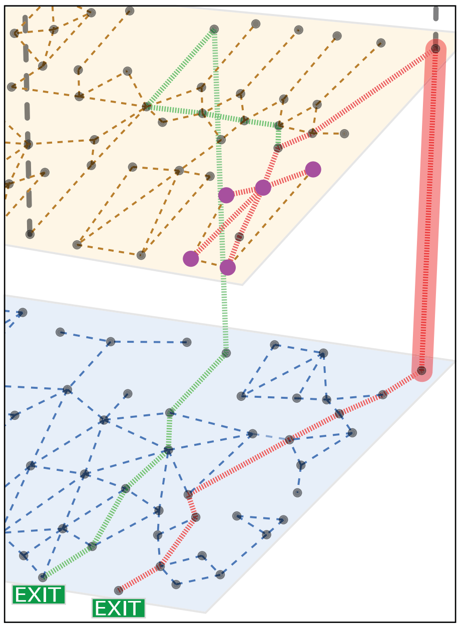


Figure 4.5: Detail of the building graph (Fig. 4.2) showing the edge whose distance is increased (bold red, eastern staircase), the shortest path before the length change (red) and the new shortest path after the red edge's length is increased (green, going through central staircase). The figure shows that there is no "trivial" solution to bypass the red edge whose cost is increased.

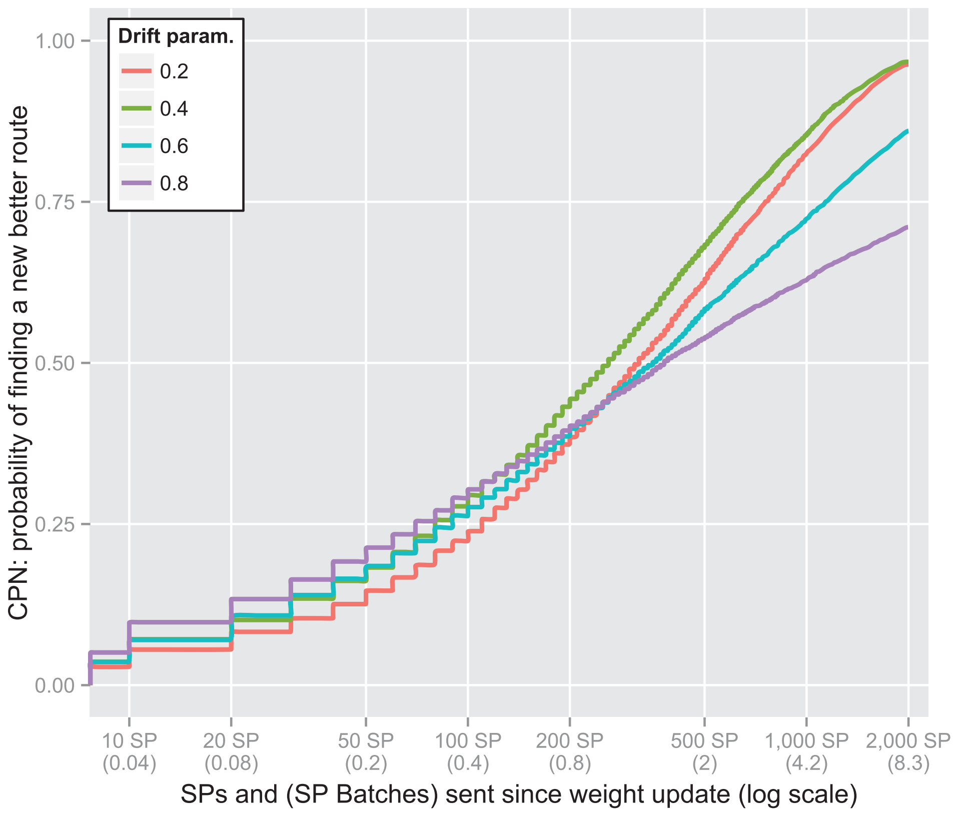


Figure 4.6: Probability of CPN switching to a better path, based on the number of Smart Packets sent after the modification in original route cost. The figure shows that high drift parameter values increase the algorithm's latency.

hop count limit is set to 50. We set the horizontal axis (number of SPs sent) on a logarithmic scale as the probability initially increases at a fast rate and generally tends to stagnate.

An attentive observation of Figure 4.6 shows the probability of CPN switching before any SP is sent is *not* null. Yet CPN cannot notice the change in red path length and start searching for alternatives without sending a single SP. This anomaly is caused by route errors: CPN has in fact *mistakenly* identified the green path as being shortest *before* we even increased the length of the red edge. This shows CPN is not an optimal routing algorithm, and that it may get stuck with sub-optimal routes. This tends to occur most often with high drift parameters, which validates our previous conclusions. Overall, this error remains very small and does not influence the rest of the experiment.

Figure 4.6 shows the drift parameter does not have a strong influence on the likelihood of switching routes before 200-300 SPs are sent. However beyond 300 SPs, low drift parameters reduce latency, and more than 95% of route switchovers occur before the 2,000 SP cutoff limit. In contrast for the highest drift parameter values, the latency is higher and more than a quarter of the routes do not switchover before the cutoff limit. Usually, the first step of a switchover process is to detect that the path metric of the red route has increased. This increase in length may go unnoticed for some time with low drift parameter values, as SPs partly ignore the RNN's advice which is to go along the best known path. This could explain why the switching probability of drift parameter = 0.2 is lowest in the early stages of the experiment: CPN takes time to realise the red route's distance has increased. However, once the red route's cost is updated, the largely random SP movement means that the green route is more likely to be found. On the other hand, where the drift parameter is high, the bulk of the SPs will be visiting the best known route (red) and almost immediately detect the variation in path length. However, SPs will continue to visit the red path until the RNNs have received enough feedback to effectively punish the neurons associated with this path. Eventually the feedback will make all neurons's excitation converge to the same value, indicating that the RNN does not have a clear solution to the problem, yet the high drift parameter coerces the SPs in following the poor advice provided by the RNNs: this may delay indefinitely the discovery of a new path. In order to validate these hypotheses, we would need to statistically analyse the path of individual SPs and monitor the changes in neuron excitation throughout the process.

Other Parameters

In comparison with the drift parameter, we found that the remaining parameters had either a trivial or marginal impact on CPN's update latency. The following is an overview of some of the trends we have identified while analysing the effect of the remaining parameters. The plots relevant to this section are in the appendix.

- **Anterior training**. After the initial route discovery process, we send an additional {10, 20, 40} SP batches before increasing the length of the edge on the red path. We notice slightly better update latencies when less SPs have been sent prior to the path length increase. This is possibly because the RNNs have received less reinforcement for the red path, which makes it faster to "overturn" the RNNs. Much like the previous point, there is insufficient detail in our experimental data to validate this theory. The relevant plot is in the Appendix section, see Fig. 8.4

- **Damping coefficient**. We expected our bench test to reveal variations in update latencies based on the damping coefficient but found no significant differences.

Setting a low damping coefficient means the threshold merely tracks the last SPs' result and fails to provide a reliable point of comparison to judge the performance of new routes and provide adequate feedback to the RNNs. On the other hand, a high damping coefficient means that the threshold will vary very slowly. Thus the RNN may continue to have "high expectations" – which the green path may not meet – until enough feedback has been received to progressively raise the threshold value above the green path's length. The fact that either extreme values of the damping coefficient may have an adverse affect on the switching latency could explain why we do not see significant variations based on this parameter value. Since the damping coefficient controls the low-pass filter behaviour of the threshold, this parameter could have a more significant impact on controlling route oscillations seen in the presence of sensitive metrics [50]. The relevant plot is in the Appendix section, see Fig. 8.5.

4.4 Summary

In this chapter we have introduced CPN, the routing algorithm we will use in this work. We have presented its advantages over classic routing algorithms: distributed operation and compatibility with a variety of metrics. By incorporating RNNs into CPN nodes, we have shown that the routing algorithm is also able to manage path discovery and update overhead efficiently, focussing on areas of the network perceived as most worthwhile.

In the second part of this chapter, we have presented a set of experiments to determine the optimal parameters for our application. To the best of our knowledge, we are the first to present such a methodical sensitivity analysis of CPN's parameters: most articles found in the literature set values through a trial-and-error process or other heuristics. We were able to describe the effects of CPN's parameters during the initialisation process, however, the depth of our *dynamic* analysis only allowed us to set parameters empirically. We could only formulate hypotheses on the underlying path update process to explain our observations.

Based on our analysis, we have decided to set the drift parameter to 0.5 to obtain a good compromise between route resolution and low latency. We set the hop limit to 45 as it seems to be a good compromise between performance and overhead. We have also set the damping coefficient to an intermediate value of 0.5.

Our evacuee guidance system may run on a pool of networked devices disseminated in the building. As such, versions of CPN tailored to the needs of ad-hoc networks may be relevant [78, 53] and should also be tested. The complexity of the calculation associated with the RNNs may also be a limiting factor, if the platforms provides very little computational power. In such cases, we recommend considering using an aRNN [80], a concept which aims at replicating the process of an RNN but uses significantly less computational resources, or routing SPs using a sensible routing policy [47].

5 Metrics for Congestion-Aware Routing

In the previous chapter, we have demonstrated that CPN is able to identify optimal paths in our building graph, with path length as a metric. While this metric could be used to route evacuees, we will demonstrate the limitation of this approach in the first part of this chapter. In order to address this limitation, we propose two path metrics which, in combination with CPN, have the potential to optimise the flow of evacuees during the evacuation. The first metric is a simple real-time path delay measurement, while the second metric is based on future capacity reservation. In the last part of this chapter, we review the implementation constraints associated with each metric, and compare their performance in simulated evacuations.

5.1 Shortest-Path Metric

Let us assess the potential of a shortest-path evacuee routing scheme by defining how well it meets the Uniformity Principle. Under the Uniformity Principle, the routing scheme should direct evacuees to every bottleneck in numbers which are proportional to the bottleneck's maximum flow. In our building, the main bottlenecks are the staircases, which have the same maximum flow values. Therefore, a Voronoi diagram of our building drawn with respect to these bottlenecks will give us an estimate of their respective catchment areas. This is shown in Figure 5.1, where the area with a blue overlay will evacuate on the eastern[1] side of the building. All other areas will evacuate the building via the central stairs and exit, under the shortest-path routing policy. We have also drawn the most-used edges using a bolder line: they correspond to the main egress paths in the building. Figure 5.1 shows evacuees on the ground floor will use both exits, with a slight preference for the central exit. The majority of first-floor departure points will transit via the central staircase and exit, whereas evacuees on the second floor tend to use the eastern staircase and exit (blue overlay). The two upper floors have the same area and evacuee capacity, and staircases have the same maximal allowable

[1]The eastern part of the building is on the right of the diagram on Figure 5.1

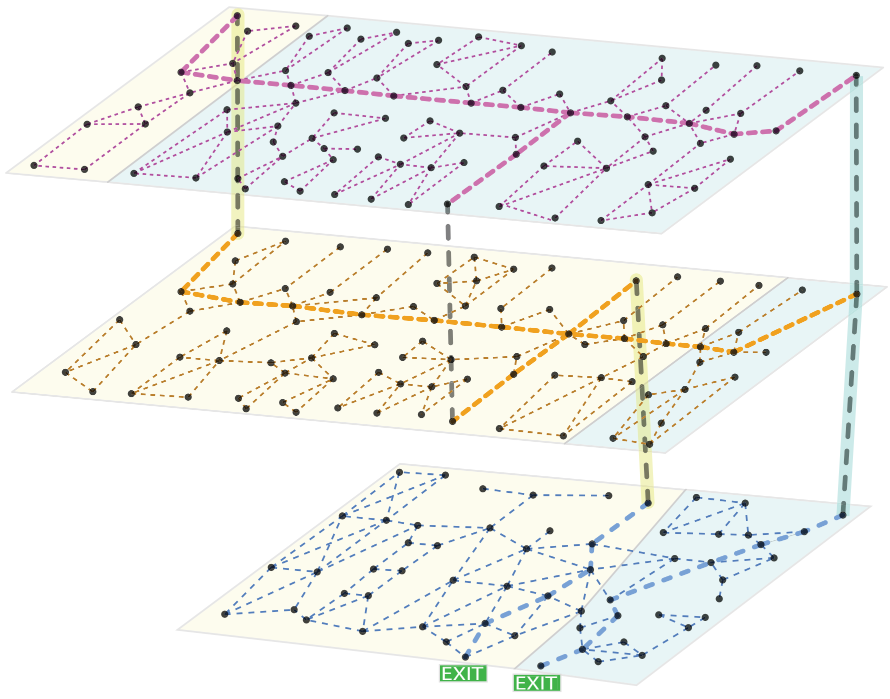


Figure 5.1: Staircase affinity of each departure point (using shortest-path). Departure points with a blue overlay use the eastern staircase, the other ones use the central staircase. Main egress paths in the building appear with bold edges. The figure shows that most evacuees on the first floor use the central staircase, while those on the second floor will mainly use the eastern staircase.

flow. As this building has been well-planned according to the "worst-case" principle, if filled to capacity, the building will be evacuated in flow-optimal conditions, with all bottlenecks (stairs and exits) equally "loaded" and therefore operating at full capacity throughout the evacuation. Let us now consider a scenario where the building must be evacuated at a time when lectures take place on the first floor, while the second floor and ground floor lobby area are virtually empty. Under these circumstances, most of the evacuees will use the central staircase, while the eastern staircase remains virtually empty. Clearly, the shortest-path routing approach is ineffective in this situation. The same logic applies if all users were on the second floor: the eastern staircase would be overloaded. This shows that individually minimising each evacuee's path length does not lead to a global optimum in all circumstances. Instead, a routing policy which makes use of all safe paths – not just the shortest one(s) – and takes into account the capacity on each path has the potential to greatly improve the evacuation time. This is especially valid in buildings such as ours, where alternative paths with less congestion are sufficiently accessible, so that diverting evacuees there will not incur a large delay.

5.2 Congestion-Oriented Path Metric

In this section, we introduce two separate methods to estimate the path traversal time metric, which are compatible with the CPN routing algorithm. Instead of solely representing path distance, both metrics aim to represent the *time* needed to walk along this path, accounting for congestion-related delays and queuing time. Our system does not *explicitly* aim at balancing loads, however, by dynamically assigning (or redirecting) evacuees towards the quickest path, the outcome, in practice, is similar to load balancing.
The first metric calculation method we introduce is based on real-time congestion measurements, which makes this approach fundamentally reactive: congestion first has to occur before it can be reflected by the path delay metric and potentially addressed by the routing algorithm. This mandates an on-line routing approach, to address the constant variations in edge travel times. The second method to estimate the path delay metric is based on a proactive approach, by using future capacity reservation to forecast congestion and estimate queuing times. We regard it as proactive since it allows the routing algorithm to compare congestion on different routes ahead of time, which allows the routing algorithm to operate off-line.
We refer to the first approach as "Reactive" and the second one as "Proactive". When using CPN in conjunction with either metric calculation method to route evacuees, we may use the terms "Reactive routing" and "Proactive routing". Both approaches pro-

posed to calculate the path delay metric require a sensing system able to determine the evacuees' location. They also require a graph-based representation of the area, where nodes symbolise physical areas in the building and edges represent paths between them. This graph should also contain information on:

- Edge distance, so that a transit time T_v can be calculated for each edge v based on the evacuees' nominal walking speed, and
- Edge's capacity C_v, defined as the number of evacuees which can concurrently travel along an edge.

5.2.1 Reactive Path Metric

The Reactive approach is fundamentally a real-time estimation. It is based upon the assumption that the queue levels in the building reach a steady-state value. Under the assumption that the queue levels are stable, a path transit time can be calculated ahead of time, based on *current* congestion measurements. For a given path P composed of a collection of edges V^P, the path transit time is estimated as follows:

$$T_P(t) = \sum_{v^P} \frac{\left(\left|F^*_{v,t}\right| + 1\right)}{C_v} \cdot T_v \tag{5.1}$$

Where $\left|F^*_{v,t}\right|$ is the number of evacuees queueing for v at the instant t, when the calculation is made. Due to the steady-state assumption associated with this approach, the algorithm will only determine optimal solutions if the conditions remain unchanged. In reality, the evacuation process is unlikely to reach a strict steady-state regime, as each routing decision will affect the transit times. Therefore the routing algorithm will need to re-run periodically throughout the evacuation and issue path updates to evacuees to divert them from congested areas, as it becomes aware of them. The existence of an underlying feedback loop between the path assignment and the increase in traffic along the route is a risk factor for the routing algorithm to develop an oscillating pattern.

5.2.2 Proactive Path Metric Estimation Method

The Proactive metric estimation approach forecasts congestion by keeping track of the increase in traffic arising from each new path allocation. The first step in this process is to discretise the evacuation time in K steps of duration Δt_S. The k^{th} time-step spans over $k \cdot \Delta t_S \leq t < (k+1) \cdot \Delta t_S$. For each edge in the graph, we determine the maximum number of evacuees which can transit through this edge within a time-step, based on the edge's transit time and capacity:

$$max(|F_{v,k}|) = Round(\frac{\Delta t_S}{T_v} \cdot C_v) \tag{5.2}$$

Since we are dealing with individuals, the value must be an integer, hence the rounding operation.

For each edge, and at each time-step we create a "time-bin". Each time the routing algorithm assigns a path to an evacuee, it puts a "token" (which symbolises a capacity reservation) in every time-bin along the path at the expected time of arrival. We denote the set of evacuees which have made reservations in the k^{th} time-bin associated to edge v: $F^B_{v,k}$. Time-bins can only accept $max(|F_{v,k}|)$ (defined in Eq. 5.2) such tokens: this enforces the maximum flow and capacity constraints of each edge. If the time-bin corresponding to the expected time of arrival is full, the algorithm considers the evacuee will have to queue, since the edge is saturated. The algorithm looks up time-bins associated to later time-steps, and adds a token in the first one it finds with available capacity, and considers the evacuee will have to wait until then before progressing further. The evacuee's departure time from the edge is determined based on the situation upon arrival:

- The first case is when an evacuee's forecasted arrival time coincides with a time-bin below capacity. In this case, the evacuee does not need to queue: the system simply calculates the transit time based on the edge's free-flow transit time T_v.

- The second case is when an evacuee's forecasted arrival time is at the k^{th} time-bin, but this one is full: $\left|F^B_{v,k}\right| = max\left(|F_{v,k}|\right)$. In this case, the algorithm looks up the $(k+n)^{\text{th}}$ time-bins $(n > 0)$ and identifies the first one with spare capacity:

$$min(n) \ : \ \left|F^B_{v,k+n}\right| < max\left(|F_{v,k+n}|\right) \tag{5.3}$$

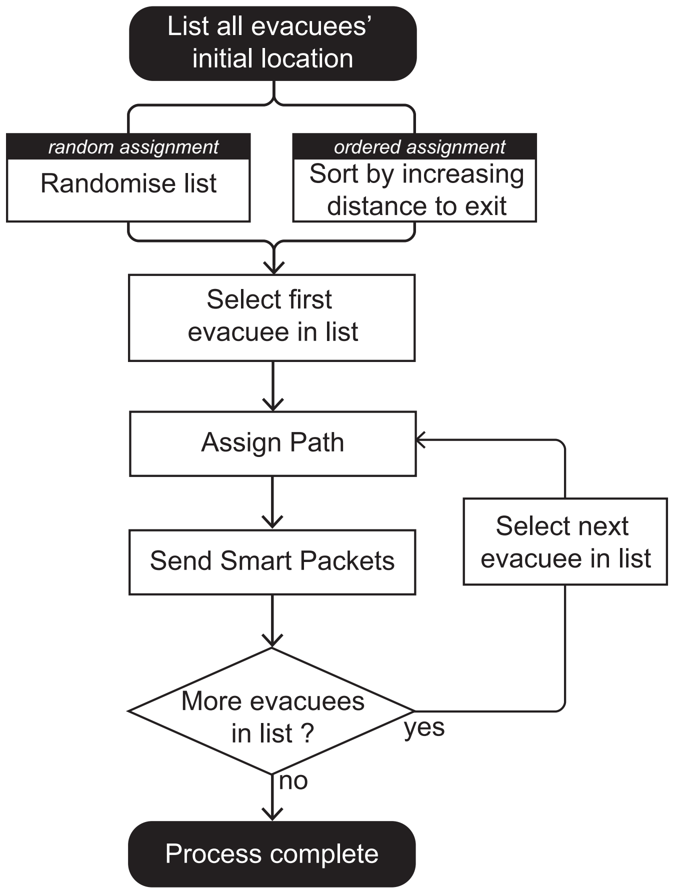
  We consider that the evacuee will be free to depart from the *beginning* of the $(k+n)^{\text{th}}$ time-step, and add the edge transit time T_v to determine departure time. We take an optimistic approach, expecting the user will be free to move along the edge from the beginning of the allotted time-bin. We could refine this by taking into account the order of the reservations made to the time-bin, e.g. the first to reserve capacity is the first to arrive and would leave first, etc. However, as we will see, the order of arrival in the time-bin is often inaccurate, and often irrelevant with sufficiently small time-steps and large numbers of evacuees.

During the path assignment process, the initial version of this algorithm does not select evacuees according to any particular priority regime: an evacuee is picked at random,

assigned to the best available path, and the process reiterates until all evacuees have a path. This can be considered as a flaw in the algorithm's model: evacuees which are closest to the exits will generally be the first to reach a particular node, and should therefore be assigned paths first. For instance, if the evacuee furthest away was first assigned a route, all time-bins would be empty (since the system has just been initialised) and the system would consider its path congestion-free. However, in practice, this evacuee is likely to be involved in queues as evacuees which are closer to the exit will arrive before him and will slow down his progression. Thus it would be more logical for this evacuee to be assigned a route last. To verify this theory, we have designed and evaluated a variant of the algorithm where routes are assigned by decreasing order of proximity to the exit.
Figure 5.2 illustrates the route assignment process and the difference between the *ordered* and the *randomised* variants. Algorithm 1 formally defines how the transit time of a given path at a given departure time is obtained. The switch on line 9 allows this algorithm to be used either to merely determine a path transit time, or to reserve capacity. The routing algorithm sets `reserve=false` to analyse paths discovered by Smart Packets and to rank them in the routing table, without making any capacity reservation. On the other hand, the flag is set to `true` when the routing algorithm reserves capacity after assigning a path to an evacuee. Each successive route assignment gradually reduces the available capacity of each edge, and transit times increase. In order to monitor these gradual changes in edge transit delay, CPN sends a batch of Smart Packets between route assignments.

```
**Data**: Departure time $T_{departure}$; Path $P$
**Result**: Path Transit Time and Capacity Reservations
1  $T_{departure} \leftarrow T_{arrival}$
2  forall the $Edges\ v \in Path\ P$ do
       /* Find earliest free time-bin                                    */
3      $k \leftarrow$ Floor($T_{arrival}/\Delta t_S$); //time-step at expected arrival
4      $n \leftarrow 0$;
5      while $k + n \leq K$ do
6          if $\left|F^B_{v,k+n}\right| = max\left(|F_{v,k+n}|\right)$ then
7              $n \leftarrow n + 1$; //time-bin full, move to next one
8          else
9              break;//k+n is now the earliest time-bin with capacity
10         end
11     end
12     if $reserve{=}true$ then
           /* Reserve capacity in earliest non-full time-bin                */
13         $\left|F^B_{v,k+n}\right| \leftarrow \left|F^B_{v,k+n}\right| + 1$;
14     end
       /* Determine departure time                                        */
15     if $n{=}0$ then
           /* Free capacity at the forecasted arrival time                  */
16         $T_{departure} \leftarrow T_{arrival} + T_v$;
17     else
           /* Evacuee must queue for n time-bins                            */
18         $T_{departure} \leftarrow (k + n) \cdot \Delta t_S + T_v$;
19     end
20     $T_{arrival} \leftarrow T_{departure}$;
21 end
22 return $T_{arrival}$
```

Algorithm 1: Capacity reservation algorithm.



List all evacuees' initial location
random assignment
Randomise list
ordered assignment
Sort by increasing distance to exit
Select first evacuee in list
Assign Path
Send Smart Packets
Select next evacuee in list
More evacuees in list ?
yes
no
Process complete

Figure 5.2: Flowchart of the Proactive route assignment process, highlighting the difference between randomised and ordered variants.

5.3 Simulations and Results

We use the DBES to evaluate the performance of the CPN routing algorithm with path delay as a metric, using the two calculation methods presented in this chapter. Before presenting the simulation results, we present our simulation scenarios and parameters.

5.3.1 Simulation Scenario and Parameters

We run a total of 320 simulations, made up of 20 randomised runs of 16 simulation scenarios formed by the combinations of two parameters.

Scenarios

- Four different initial evacuee headcount $|F| \in \{25, 50, 75, 100\}$.
- Four different algorithms:

1. **Reactive**: CPN routing algorithm with path delay metric, Reactive estimation.
2. **Proactive, ordered**: CPN routing algorithm with path delay metric, Proactive estimation. Paths are assigned with a priority regime based on distance to the exit.
3. **Proactive, random**: CPN routing algorithm with path delay metric, Proactive estimation. paths are assigned in a random order.
4. **Shortest path**: Static approach where evacuees follow the shortest-path to the nearest exit. This serves as a control test as it represents the evacuee's instinct to reach for the nearest exit.

Randomisation

In order to obtain a representative value of the "general" performance of both metric estimation methods and routing algorithm, we run each scenario twenty times, with some parameters randomised:

- **Initial location** At the start of each simulation DBES assigns to each evacuee an initial location chosen at random from a set list of possible departure points.
- **Departure time** It is well known that evacuees do not start evacuating as soon as the alarm goes off (see, for instance [106, 107]). Therefore DBES holds evacuees in their initial place for a moment before letting them make their way towards the exit. This delay is assigned at random, according to a Gaussian distribution of mean $\bar{t} = 5$ and standard deviation $\sigma = 5$. We use absolute values to avoid negative values.
- **Smart packets** As we have highlighted previously, SPs are allowed to "drift" at random. The Random Number Generator used to decide whether or not to drift, and if so in which direction this should be done, is initialised with a different seed at each run. The next node to send an SP is also chosen at random according to a uniform distribution, so that over the course of the simulation every node sends approximately the same amount of SPs.

Initial Evacuee Locations

We have previously demonstrated that our building is designed to produce near-optimal flows under the shortest-path routing policy (section 5.1), when filled to capacity. Therefore, there is little use in testing our system under a full-capacity scenario, as a simpler shortest-path algorithm is sufficient. Our objective is to design a system which minimises building evacuation times *regardless* of the number of evacuees and their initial distribution. Therefore, we purposefully generate an uneven distribution of evacuees to challenge our routing system in cases where the shortest-path approach would be inefficient. For this purpose, we configure DBES so that evacuees only start from the first floor. The Voronoi diagram (Fig 5.1) shows that under the shortest-path routing approach most evacuees would transit through the central staircase and form high levels of congestion in this area while the second (eastern) staircase would remain virtually unused.

Sensing and Control

The aim of this section is to validate the effectiveness of the CPN routing algorithm and the two path metric calculation approaches which we have introduced. To focus on the routing component, we assume that the system's other components are *ideal*. In particular, we consider that the system is able to accurately sense the evacuee's location, is able to communicate the designated route to each evacuee, and that the evacuees strictly follow their assigned route. We will relax some of these assumptions in the next chapter, once the routing component is tested.

Duration of a Time-Step

To have the highest possible resolution in the most critical area of the graph (the staircases), we set the time-step duration (Δt_S) to match the transit time of the staircases. We have chosen this value to avoid round-off errors on the staircases: this allows the algorithm to precisely forecast the load on each staircase, which is a critical part of the routing problem. A rounding error may be introduced while calculating the time-bin capacity of each other edge. While we expect these errors to remain small, the model's accuracy could be further improved by keeping a running total of rounding errors, and each time the value reaches one, extending the capacity of the corresponding time-bin by one. For instance, if the capacity of time-bins for a given edge is 10.5 evacuees, we would create a series of time-bins with capacity alternating between 10 and 11.

5.3.2 Analytic Solutions

Before presenting our simulation results in the next section, let us apply some of the principles we have introduced in the Background chapter (Chp.2) to analytically determine the results we can expect from our simulations.

Uniformity Principle

As we have written earlier, our experiments consider scenarios where evacuees are all located on the first floor. In order to reach the exits on the ground floor, evacuees must use either the central or the eastern staircase, which are the main bottlenecks in our building graph. Indeed, the staircases are the edges with the lowest capacity and longest processing time; both of which are identical for every staircase in the building. The Uniformity Principle says that the building evacuation time will be minimal if all bottlenecks are saturated and clear their last evacuees at the same time. This is valid if evacuation paths are equally accessible and do not interact. Our scenario and building graph happen to validate these assumptions quite well, since both staircases can be considered as accessible. That is, following a path through either staircase does not significantly change the path length, and these paths are mostly segregated from one another. To realise the Uniformity Principle, we must assign evacuees to each staircase in numbers which are proportional to the staircases' maximum allowable flow. Since the staircases have identical parameters, the solution is simply to divide the evacuee population in two, and assign each half to a separate staircase. We introduce the term "staircase balance": as the ratio of users which use the central and eastern staircase. The usage of the staircases is *balanced* if the number of evacuees travelling through them is equivalent (1:1 ratio), and *unbalanced* otherwise.
We empirically verify that this solution is indeed correct by computing the staircase balance for each experiment conducted as part of this work, and comparing it to the average individual evacuation time. Thus each of the 1000 points on Figure 5.3 correspond to one of the simulations we have run. The horizontal coordinate is the staircase balance, and the vertical coordinate is the average individual evacuation time. It is clear from this figure that the individual average building evacuation time reaches a minimum when the usage of each staircase is balanced. According to the Triple Optimisation principle, this means that the overall building evacuation is also minimal when the staircase assignment is balanced.

Lower-Bound Evacuation Time Estimation

We have seen that our particular evacuation scenario uses two main evacuation paths, and we used the Uniformity Principle to determine the optimal assignment of evacuees on each path. Using the equation related to the transit time of F units through a capacitated path P (Eq. 2.2), we can determine a lower-bound estimate on the building evacuation time, for any evacuee headcount.
We calculate the path lead-time by computing an average of the shortest-path to the

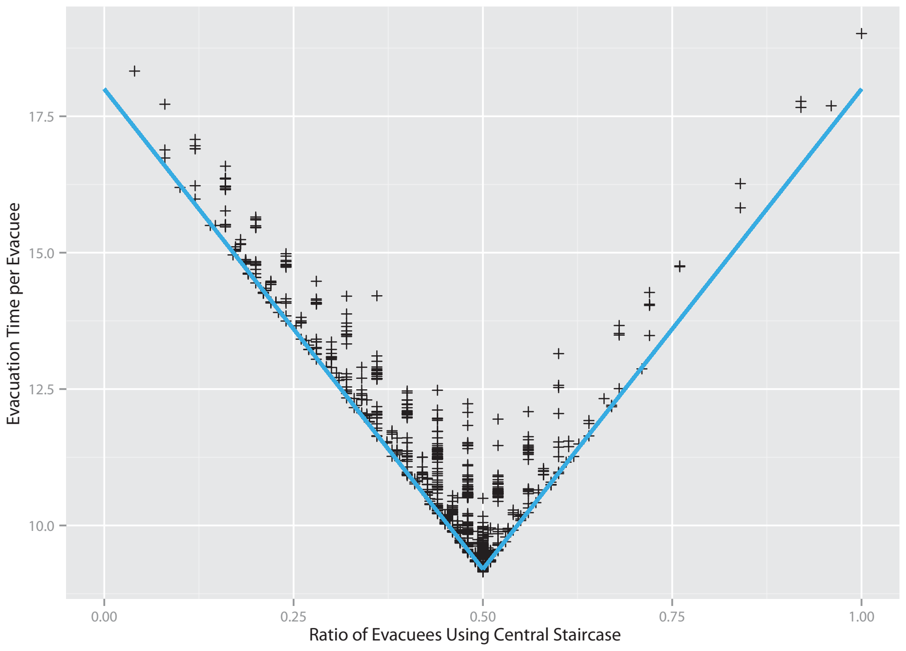


Figure 5.3: Normalised evacuation time based on the ratio of evacuees which have used one of the two staircases (in this case, the central one). Data extracted from over a thousand various experiments. The graph shows that the minimal individual evacuation times are obtained with staircases distribution ratios of 1:1 (0.5)

exit, for evacuees evenly distributed on the first floor. Since each staircase should ideally process one half of the evacuees, the transit time through either staircase is calculated as follows:

$$T_{stair} = \frac{|F|}{2} \cdot \mu_{staircase} \tag{5.4}$$

We show the results of this calculation on table 5.1, and also display them on evacuation time plots.

	25 Evacuees	50 Evacuees	75 Evacuees	100 Evacuees
Min. bldg. evac. time	260 sec.	480 sec.	700 sec.	920 sec.

Table 5.1: Lower bound evacuation times determined analytically, using the Uniformity Principle and Equation 2.2.

5.3.3 Simulation Results

Figure 5.4 shows the distribution of results from the simulation runs. The horizontal lines mark the lower-bound building evacuation time calculated previously.

Improvements over Shortest-Path Routing

As expected, the SP routing algorithm performs poorly since users are not distributed evenly across the building. The improvement brought by managing congestion ranges from 27% for the smallest evacuee population to 33% for the largest one. We also notice that the congestion-aware algorithms' evacuation times are very close to the lower-bound estimation of the building evacuation time. Figure 5.5 shows the time at which each evacuee walks past the top of either staircase on the first floor, on the way to the ground floor. In shortest-path routing only 20 evacuees use the eastern staircase while the remaining 80 use the central staircase, which means the eastern staircase's capacity is unused during nearly three quarters of the evacuation time. As a result, the overall building evacuation time is one third longer compared to cases where the staircase loads are adequately balanced.

Comparison of Variants of the Proactive Algorithm

We proposed and simulated two variants of the Proactive algorithm, which differ in the order in which evacuees are assigned a path. We recall that:

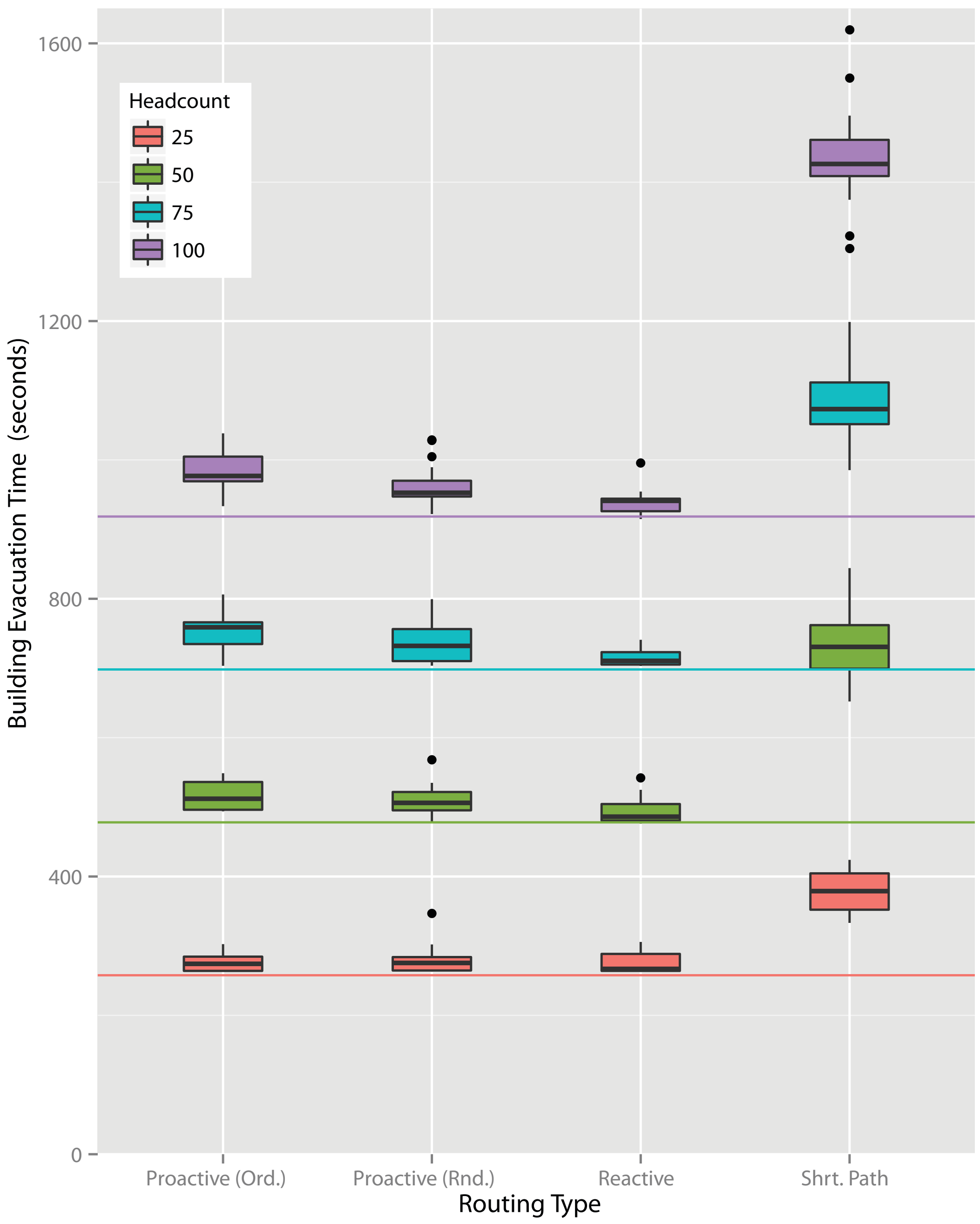


Figure 5.4: Building evacuation times: experimental results and lower bounds. The graph highlights the poor performance of the shortest path routing approach when evacuees are not distributed evenly in the building. We can see that the proposed routing metric reaches near-optimal performance levels, regardless of the estimation method.

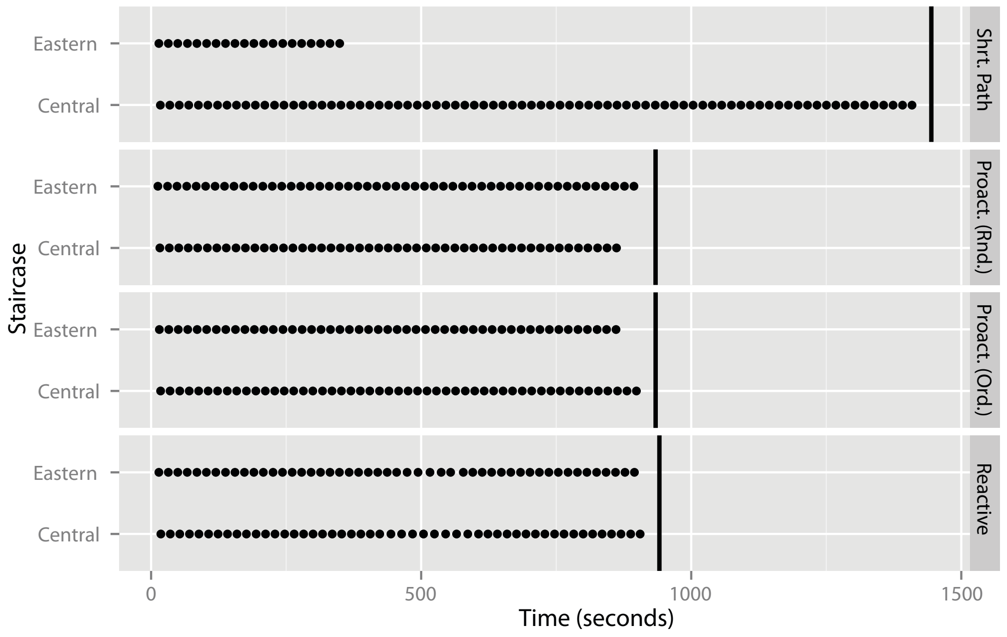


Figure 5.5: Time of passage of evacuees at both first-floor staircases: each evacuee corresponds to a dot. Based on single, representative simulations featuring 100 evacuees. The thick black vertical lines show the corresponding building evacuation time. This schematic highlights the poor use of the staircases' total capacity with shortest path routing. In contrast, our routing approach distributes the evacuees so that both staircases are used at their full capacity, for almost the entire evacuation

- In the standard or *randomised* version, routes are assigned to evacuees in a random order,
- The *ordered* version first ranks evacuees by the shortest distance to the nearest exit, and assigns paths starting from the top of that list.

We created the ordered variant to rectify a perceived flaw in the model, and thus expect it to perform better, as it is thought to build more realistic forecasts. However preliminary results did not meet our expectation: the results of the ordered version of the algorithm were far worse than those of the randomised one.

We traced the root of this problem to CPN: the routing algorithm. Let us recall that CPN sends Smart Packets from each node after each route assignment to monitor the changing state of the network, as capacity gradually decreases after each assignment. As evacuees are ordered by distance to the exit, evacuees *in the same area* get assigned a path all at the same time. This results in a sudden decrease in available edge capacity in this particular area. As the SP graph update process is gradual and diffused, CPN is unable to keep up with the fast and localised changes in edge transit delay and update its routing tables in time. In contrast, when we assign paths to evacuees chosen at random, the changes in edge transit times are much more gradual and diffused over the building graph, and very few SPs suffice to handle these variations and consistently find optimal paths.

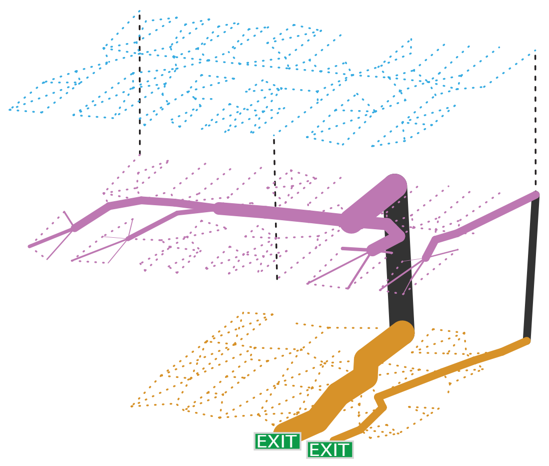
In summary, the ordered version causes sudden and localised variations in edge delays that "overwhelm" the routing algorithm. To remove the routing algorithm's influence, we have allowed CPN to send additional SPs between each ordered route assignment, which effectively reduces the algorithm's latency. We have done so until both variants reached equivalent levels of performance, and used this as our final result on Figure 5.4. Our conclusion is that, while sending one SP from each node after every route assignment is enough to reach near-optimal results with the *randomised* route assignment scheme; this needs to be increased to 10 for the *ordered* variant to reach the same level of performance. Therefore, while Figure 5.4 shows somewhat comparable performance levels between the two variants, the ordered variant incurs a significantly higher computational cost due to the tenfold increase in SPs.

Beyond concerns related to routing algorithm's performance, Figure 5.4 shows that the randomised version's results are close to the optimal solution anyway. This suggests that the randomised variant performs well despite its flaw; and perhaps the ordered variant only adds unnecessary detail and complexity, in this case. We attribute this to the fact that all bottlenecks operate in saturated regime throughout the evacuation, that is, there will always be a queue formed ahead of the staircases. In these conditions, it makes little difference if an evacuee arrives later than the time-bin he had originally reserved, as another evacuee ahead of him in the queue will honour his reservation. This explains why the randomised algorithm performs well in our scenario, and probably in most evacuation scenarios where the number of evacuees is sufficiently large to saturate the bottlenecks.

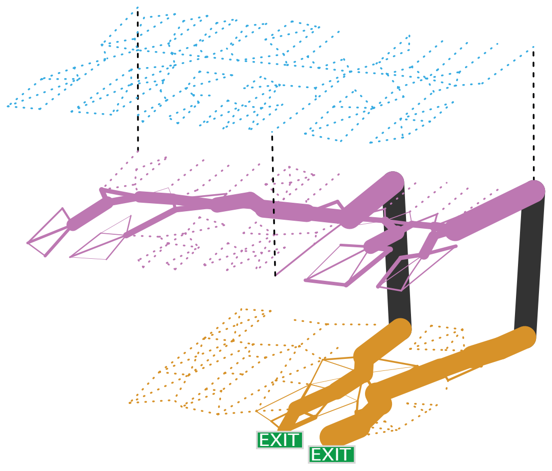
Comparison of Reactive and Proactive Algorithms' Results

Let us now compare the performance of the Reactive and Proactive (randomised route assignment order) algorithms. Figure 5.4 shows that both algorithms have comparable results in terms of building evacuation times. Beyond minimising evacuation times, we are also concerned with the "quality" of the path received by evacuees: is it rational, straightforward and not excessively long? There is no obvious metric to comprehensively determine the "cost" of of a path, especially from an evacuee's point of view. Our routing algorithm only attempts to minimise path transit times, however, walking *against* the main stream of evacuees could be considered as being more difficult than moving *along* with the flow. Likewise, a path composed of several turns should have a higher cost, compared to one which is more straightforward, as it requires the evacuees to pay closer attention to the signs. Any detour should also have a higher cost, not merely because of the increased walking distance, but also because taking a detour, from the evacuee's point of view, is not *intuitive*. Assigning an evacuee on a detour path requires a certain level of trust in the system from this individual.
Clearly, the quickest path may not always be the best, and an ideal metric would also compound length, simplicity, intuitiveness, and more. Our first step, to analyse the path "quality", will be to consider its length. While we will not elaborate a *quantitative* metric which accounts for all the human factors, we will however extract some sample paths followed by actual evacuees and *qualitatively* analyse them.
Figure 5.6 gives us an insight in the the path usage statistics: the line thickness corresponds to the average number of visits[2] received by each edge in the graph for each routing type[3]. While Figures 5.6b and 5.6c appear similar, there is one notable exception: the edges on the first floor, between the two staircases, are much more active in Reactive routing than in Proactive. In Proactive routing, evacuees are source-routed: they receive an optimal path, which is *not* corrected later on. On the other hand, the Reactive algorithm reacts to imbalances in bottleneck loads by sending route corrections to evacuees which diverts them off congested paths. Thus the edges between the two staircases are used more often because the Reactive algorithm has to "shift" users from

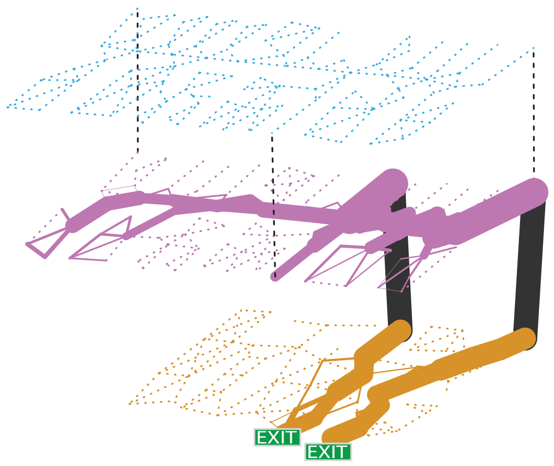
[2]If an evacuee visits the same edge several times, each visit is counted.
[3]The results of the variants of the Proactive routing (ordered and randomised route assignment) are amalgamated, as they are very similar.



(a) Shortest-path routing.



(b) Proactive routing.



(c) Reactive routing.

Figure 5.6: Edges visited during evacuations featuring 100 building occupants. Line thickness is proportional to the number of visits. We can see that Proactive and Reactive achieve good load balances on staircases. The thickness of the edges *between* both first-floor staircases is Reactive routing reveal the presence of routing oscillations.

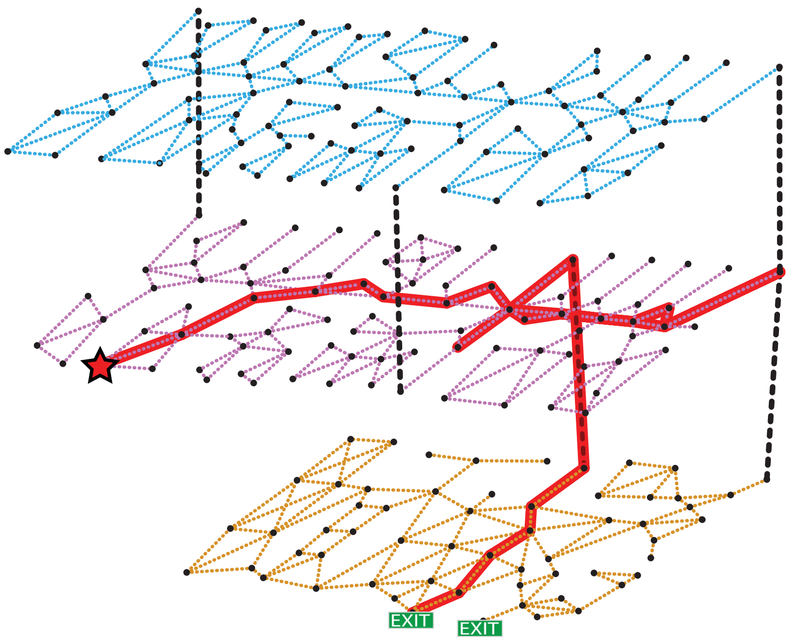


Figure 5.7: Trace of the evacuation path (bold line) followed by an evacuee routed using the Reactive algorithm. The evacuee starts from the leftmost location on the 2$^{\text{nd}}$ floor marked by a red star. The path contains loops, is needlessly long and clearly incoherent from an evacuee's perspective. This is the result of routing oscillations.

a staircase to another, in order to maintain the balance of loads between them. Figure 5.7 is a typical example of an individual's evacuation path routed using the Reactive algorithm. This confirms that the evacuee is instructed to go back and forth between the two staircases, and suggests that the algorithm is oscillating.

In data networks, routing algorithm oscillations are often undesirable: they cause packets which originate from the same source to take different paths, and often arrive out of sequence. Yet routing algorithm oscillations can also maximise the throughput [49]: In fact, if the routing algorithm did not change routes at all, all evacuees starting from the same node would follow the same route. This route would be overloaded, while other routes could be left unused, leading to a sub-optimal flow distribution.

While oscillations contribute to load-balancing, excessive oscillations can also direct evacuees in a back-and-forth motion shown by Figure 5.7. This pattern is due to the real-time operation of the Reactive algorithm, and because it ignores the existence of a delayed feedback loop between route assignment and congestion. Typically, the algorithm will first determine that a bottleneck v_A has become overloaded, and that paths going through another bottleneck v_B are now faster. Based on this observation, the algorithm issues route corrections to evacuees queuing for v_A, diverting them towards v_B. Since the algorithm does not factor in the available capacity in either v_A or v_B, it is unable to determine the appropriate number of evacuees which need to be diverted, and diverts them all by default. Thus, the massive departure of evacuees from v_A and their arrival into v_B reverses the situation: v_B is now overloaded and v_A's load is now below optimal. The algorithm, again, has to divert evacuees to address this situation, and it is easy to see how this phenomenon can carry on oscillating between these two states, as long as there are evacuees queuing in front of the bottlenecks. The result is that evacuees are led to walk back and forth from a bottleneck to another, as the algorithm uses them as "adjustment variables" to even the loads of each bottleneck.

Oscillations Damping Measures for the Reactive Algorithm

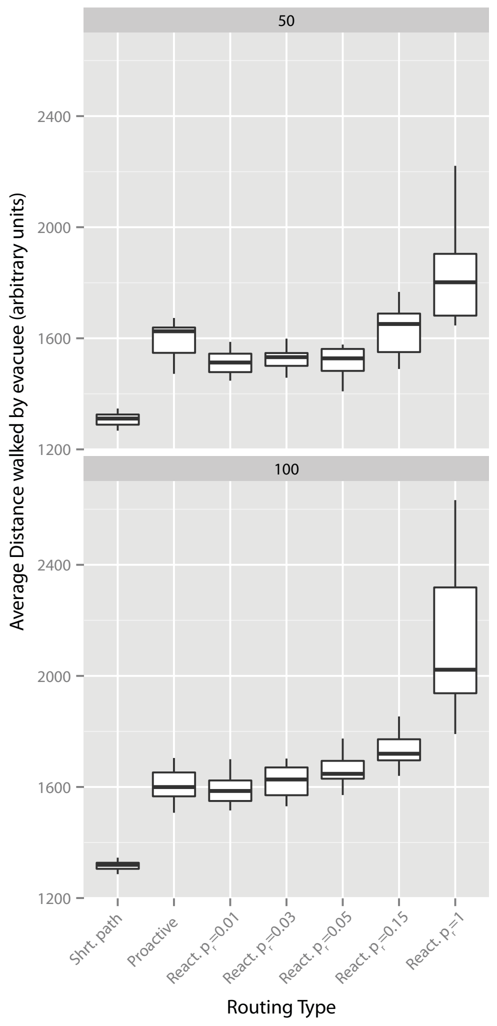
We have explained that, while route oscillation can be beneficial, in the case of the Reactive algorithm they result in unnecessary motion from a bottleneck to another without any effective progress being made towards the exit. This form of oscillation can be countered by [50]:

1. Only issuing a route update if the new route is significantly better than the previous one, using a defined improvement threshold.
2. Only issuing a route update if the user has travelled down a set minimum number of edges on the previous path recommendation.
3. Only issuing a route update to a restricted set of the population, using a probabilistic assignment scheme.

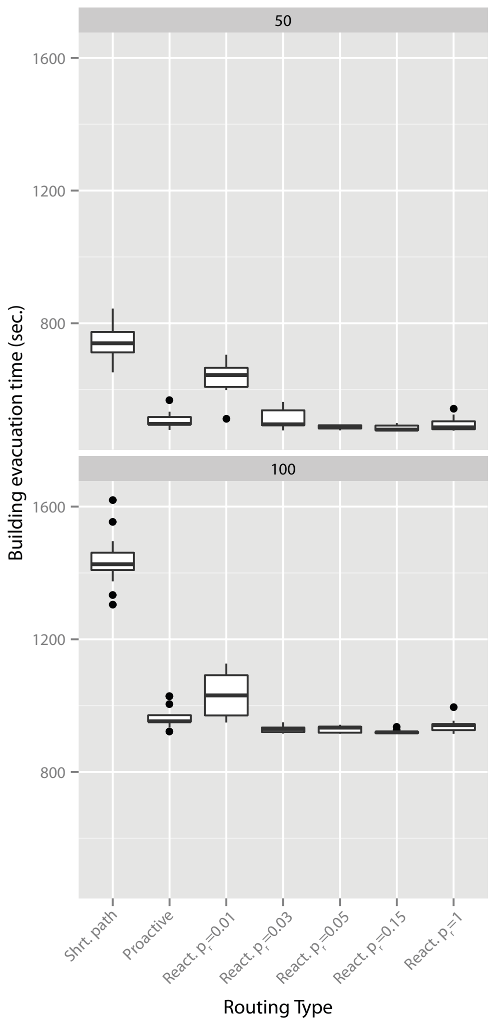
In this section we trial the last method (3.): our objective is to reduce the distance walked by evacuees *without* significantly increasing the overall evacuation time. We modify the Reactive routing algorithm so that route updates are still computed at the same time interval, but are only transmitted to an evacuee with probability p_r. Thus in the original algorithm $p_r = 1$, and we will now decrease this probability.

Figure 5.8a shows the average distance walked by an evacuee, for 50 and 100 evacuees[4] and $p_r \in \{0.01, 0.03, 0.05, 0.15\}$ and 1, i.e. the original algorithm without any oscillation countermeasure. We also show the results of the shortest-path and Proactive algorithms (on the left) for comparison purposes. The chart shows a steady decrease in evacuation path length: as the probability of issuing a route update lowers, evacuees are less likely to receive a path update and tend to follow a "straighter", shorter path. Figure 5.8b

[4]The results for a building containing 25 and 75 evacuees follow the same trends and are therefore not displayed.



(a) Average distance walked by evacuees.



(b) Average individual evacuation time.

Figure 5.8: Individual evacuation times and exit path length. While decreasing p_r continuously decreases the distance walked by evacuees, past a certain point it increases the building evacuation time.

shows that reducing the back-and-forth movement of evacuees does not increase the evacuation time: instead of constantly switching from a queue to another, evacuees simply tend to remain in the same bottleneck queue or change only once if necessary. There is however a limit, where further reducing p_r starts to increase the building evacuation time. To explain this, let us consider the extreme case where $p_r = 0$: evacuees never receive route corrections, they simply follow the route assigned to them at the very beginning of the evacuation. Yet at that time, congestion has not yet formed and the optimal solution is equivalent to shortest-path routing. Thus as we gradually reduce p_r, the path taken by the evacuees tends to resemble the (suboptimal) shortest-path solution. This explains why decreasing p_r below 0.05 for 50 evacuees (or 0.03 for 100 evacuees) actually increases the evacuation time, and by further lowering p_r, we expect the results would converge towards those of the shortest-path routing.

In summary, probabilistic route assignment is an effective solution to control excessive path oscillations: given the right parameter p_r, it outperforms all other algorithms both in terms of evacuation time and average path length. Yet we have seen that the performance is highly sensitive to p_r: too high, the average path length increases; too small, the evacuation time increases. We have also seen that the optimal value of this parameter depends on the number of evacuees in the building, and other than empirically, it seems difficult to determine the right p_r. In conclusion, while effective, this solution is somewhat impractical.

The two other oscillation damping measures we have mentioned at the beginning of this section are also parametric, and we expect the conclusions to be the same: their effectiveness will depend on how well their parameter has been configured, and there are no apparent methods to determine the optimal value of this parameter, other than empirically.

5.4 Implementation

Beyond differences in performance highlighted by the simulation results, both Proactive and Reactive metric estimation methods differ by their sensory input requirements, ability to adapt to changes, robustness, potential for decentralisation and more. In this section, we discuss the advantages and limitations of each method with regards to their implementation into a real-life evacuee guidance system.

Sensory input requirements Both metric estimation methods require knowledge of the evacuees' locations, but neither requires evacuees to be *tracked*, that is, the sensing system does not need to identify evacuees. The Proactive approach only requires the *initial* positions, while the Reactive approach requires a constant stream of measurements throughout the evacuation process. In this regards, the Proactive approach has an advantage, as it is less reliant on sensory input. Furthermore, evacuee location measurements are more likely to be accurate just before the evacuation, as evacuees are more likely to be stationary and dispersed than during the evacuation. The proactive approach is also not vulnerable to failures in the sensory system, which are more likely to occur during the evacuation as the hazards which have triggered the evacuation may damage the building's infrastructure.

Distributed implementation We chose CPN as our routing algorithm mainly because it operates in a decentralised and distributed manner. Distribution allows our system to scale to large buildings, for instance by dedicating a server to each section of the building. While it is a fact that CPN can be distributed, we must also verify that the path metric estimation methods we proposed in this chapter can be calculated in a distributed manner.
The reactive approach relies on each node visited along the SP's path to measure and add local edge delays, making it a distributed process. On the other hand, the proactive approach requires capacity to be reserved at each node on the path assigned. The path allocation process can be distributed by putting each node in charge of assigning a path to its *local* evacuees. After each path allocation this node reserves capacity by sending *requests* to all nodes on the path. A node which receives a capacity reservation request updates its time-bins accordingly. However, this decentralised approach is vulnerable to synchronous reservations: multiple nodes may simultaneously determine that a particular time-bin is under capacity and proceed to reserve capacity within it. The target node may then receive more capacity reservation requests than it is able to accommodate. Centralising the capacity-reservation process may slow it down and will create a potential single-point failure component. A distributed solution to this problem consists of notifying of failed reservation requests, and force contending nodes to apply a random back-off delay before the next attempt.

Robustness If a computing device hosting CPN nodes fails, the nodes it hosts will no longer be explored by SPs, and paths going through these areas will disappear from the routing tables [103]. Based on this observation, dividing the building into smaller areas and assigning each of them to different physical host computers reduces the extent of "unavailable" nodes for each hardware failure. This division means that an increasingly large number of computing devices will have to fail before any node becomes completely disconnected from the building's exits, which increases the system's robustness.
Beyond this, it is clear that hardware malfunction is more likely to occur during the evacuation, as the hazard (smoke, fire, explosions, etc.) may damage the building's infrastructure. With respect to this consideration, the Proactive algorithm has an advantage since it only runs at the beginning of the evacuation. In contrast, the Reactive algorithm must remain online during the entire evacuation to issue route corrections, which means it will be more vulnerable to component and network failures.

Reaction to unforseen changes While the Proactive algorithm's off-line operation is a desirable feature to build a robust system, it lacks flexibility. If the evacuee motion model is not representative of the evacuee's motion (e.g. some evacuees walk significantly faster/slower than expected), the capacity reservations will not be met, and the routes built on this information will not be optimal. If the building graph's features change, e.g. an area is made inaccessible by the spread of fire, the initial solution becomes invalid. Even if only a few evacuees need re-routing, a complete new set of routes must be calculated from the bottom up for every evacuee. Indeed, as the system reserves capacity for each evacuee at their estimated time of passage, changing a single reservation will have a knock-on effect on every subsequent capacity reservations. In contrast, the reactive approach is much more suited at handling modifications to the graph. For instance, if an edge become hazardous, the system can be set to artificially increase its transit time, and the next set of route corrections will start to offload evacuees from hazardous routes.

Towards a hybrid system We have seen that both systems have advantages and drawbacks. The Proactive approach relies on a model which confers stability, while the Reactive algorithm's oscillations can be detrimental. The Proactive algorithm requires very little information, but in turn is heavily reliant on an evacuee motion model. Finally, the Proactive routing algorithm is inflexible; while the Reactive algorithm requires frequent measurements, but can adapt to unforseen changes in building graph topology. Perhaps a hybrid system could offer the best compromise between all of these conflicting constraints. For instance, an approach based on the Kalman filter [85] could be applied: the Proactive approach would be used to calculate a stable and optimal set of routes, while regular measurements would ensure the reservations on which the Proactive algorithm's solutions are based are met. If the system detects that the congestion forecast is gradually drifting away from the ground truth, some fresh measurements could be used to "correct" the system. Yet, as we have mentioned, it is unclear how the Proactive routes can be corrected without computing a new solution "from square one".

5.5 Summary

In this Chapter, we have presented two approaches to estimate the path delay metric which enable the routing algorithm to monitor and manage congestion, in order to minimise building evacuation times. We have evaluated their performance through simulation and compared their performance and critically reviewed each method's requirements and constraints. We expect our proposed routing component to perform as well as a static shortest-path evacuation, when the building is either filled to capacity, or evacuees are evenly distributed. In this case, our system can be regarded as being somewhat "unnecessary" as load-balancing is not required in the first place. However,

outside of the aforementioned scenarios, we have demonstrated that our system greatly outperforms the static shortest-path approach, especially when evacuees are not evenly distributed in the building.
Yet we have highlighted fundamental differences between the two metric estimation methods: the Proactive one has very few requirements, but is inflexible and reliant on a model. The Reactive one can oscillate and requires frequent updates on the distribution of evacuees, but does not rely on any model and is able to adapt to unforseen changes in the building. As a continuation of this work, we have suggested a hybrid system based on the Kalman filter concept which combines the Proactive approache's stability with the Reactive one's adaptability. Additional work is required to determine a computationally efficient method to make corrections to the Proactive routes.
Finally, conducting a sensitivity analysis on input information and the motion model would determine up to which point the system is able to cope with imprecise evacuee distribution measurements, or large discrepancies in evacuee walking speeds. This would be useful to characterise the system's performance in realistic conditions.

6 Evacuee Control System

6.1 Directing Evacuees

In order to validate the routing algorithm, we have assumed that a “control and sensing” system was paired with the routing algorithm to localise users and direct their motion. Having demonstrated the routing algorithm’s effectiveness in the previous chapter, we now consider practical means of informing the evacuees, potentially unfamiliar with the building topology or the best path to take. The advantages of using a device such as a smartphone are compelling: since CPN performs source routing, the device could simply download and display the path assigned to the evacuee. Yet we argue that this solution is impractical for several reasons. Instead, we consider the use of dynamic exit signs (whose direction can be controlled) to guide the evacuees. In this chapter, we present algorithms which control the exit signs based on the output of the routing algorithms presented in the previous chapter. While this eliminates the need for every evacuee to carry a communication device, it results in a change of paradigm: the fact that signs do not discriminate users and are seen by all evacuees in the same area means we no longer have a fine control over the trajectory of each individual evacuee.

6.1.1 Approaches

Currently, evacuees are directed on a hop-by-hop basis by static exit signs and designated fire wardens. Fire wardens often receive only basic training in emergency and hazard management, and only facilitate a predetermined evacuation plan: they are not in a position to optimise the evacuation – unless given clear instructions. The same applies to exit signs: they are static and point in a predetermined direction, which may not be the best one in particular circumstances. In the following paragraph, we list the key aspects of potential approaches in providing evacuees with the *dynamic* evacuation guidance which is computed by routing algorithms.

Information Medium

The information can be provided by devices carried by each evacuee, or by devices which are part of the building’s infrastructure:

- **Individual device** Every user can carry a communication device which receives the routing algorithm's recommendation and displays it to the user. For instance, a smartphone could fulfill this task. The advantage of this approach is that evacuees can be individually routed, and if required, the device may also provide localisation information. However, expecting a handheld device to guide users during emergency evacuations is somewhat unrealistic in many aspects. First of all all, evacuees may not own such a device, and if they did, they may forget to take it (or to consult it) during the evacuation. The device could also be discharged, not updated or not compatible. Having to regularly look at the screen of a device while navigating a congested and cluttered area may divert the evacuees' attention from their environment, which is fundamentally unsafe. Furthermore, the device could easily be dropped on the ground and damaged during a scramble; and by trying to pick it up from the ground, the evacuee could get crushed by oncoming evacuees. From a behavioural point of view, the fact that each user is routed individually through a different path goes against the natural instinct to follow other evacuees and makes the fire warden's task of "herding" the crowds virtually impossible.

- **Environment based** Evacuation guidance can also be integrated into the environment, using loudspeakers or signs. The main advantage of this method is that users will inevitably hear or see the information as they scan their environment for exit paths or hazards. It also ensures all users standing at the same location receive the same information, which lets users form a group and travel together in the same direction. Fire wardens will be able to facilitate the evacuation by reiterating and enforcing the information displayed. This type of implementation also has some advantages in terms of robustness: fixed displays can be wired with a greater level of redundancy (power and information supply) and hardened against likely hazards (high temperature, water ingress, etc.).

Representation

Common wayfinding methods involve a map or a series of directions to take at each intersection. A map requires some form of processing and interpretation: this may be difficult for evacuees which are not familiar with the building, and made even harder if they are under pressure to evacuate the building quickly, and panicked to a certain extent. In contrast, showing a series of turns reduces the information to bare minimum, and breaks down the path processing task into smaller regular steps. However, evacuees can easily get lost if they inadvertently take a wrong turn and the series of turning indications on their device does not update accordingly. In this regard, a map is more

robust as it can help lost evacuees to localise themselves and find their way back.
The device's ability to sense location and orientation can reduce the potential for human error in processing the information displayed, but errors may be introduced by these sensor systems instead. On the other hand, the orientation and location of static displays is known and does not change: this means the signs can be set to point in the physical direction evacuees need to take. This removes all complex information processing task, is not vulnerable to localisation or orientation errors, and as long as the signs are deployed densely enough, evacuees cannot get lost.

Extent of Information

The information displayed to evacuees can range from a complete end-to-end path, or simply the next turning direction. Displaying the entire path allows the evacuees familiar with the building to identify a set of key waypoints they should go through and memorise their path. On the other hand, displaying small pieces of information in a timely manner, such as the direction to take at the next intersection, requires very little processing from the evacuee, and lets them mainly focus on their environment.

6.1.2 Dynamic Signs

Based on this comparison, an environment-based, hop-by-hop evacuee information system appears as the most suitable solution, especially in terms of robustness, safety and user-friendliness. This motivates our attempt to develop a system which uses this mode of guidance, in accordance with the output of the routing algorithms presented earlier. A wide range of devices can fulfil this task: loudspeakers issuing vocal commands or other sound signals, floor-mounted lights which light up a path, or roof-mounted dynamic exit signs whose pointing direction can be controlled. Most of these solutions perform the same role and are interchangeable, to a certain extent. We arbitrarily choose to use visual signals, in particular exit signs, since most evacuees around the world are familiar with these, and they are simple and inexpensive. In order to display dynamic information, we consider that the direction displayed by the signs can be modified, i.e. they are in fact *dynamic* exit signs. Dynamic signs have been the focus of some recent research projects, in particular on their effectiveness [96] and their integration into building controls systems [110].
The concepts we present are developed on two basic assumptions:

- Users always follow the direction of the exit signs. As it is reasonable to expect evacuees to distrust a sign whose direction changes too often or appears inconsistent, we monitor and attempt to minimise the rate at which dynamic signs change direction.

- By positioning the dynamic signs slightly *before* the queuing area, we expect that once evacuees reach the queue, they will no longer see it and will never change queues, even if the sign points towards a different direction.

6.2 Reactive Routing Using Signs

The reactive algorithm performs source routing, but also issues "route updates" at regular intervals to maintain the balance of loads in key areas of the building. In practice, evacuees receive route corrections which correspond to the local node's quickest known path. Upon reception of a route update, each evacuee drops its previous route and follows the newly-assigned one.
The dynamic exit signs we wish to use mandate a hop-by-hop routing approach: in this section, we present an algorithm which preserves the fundamental concept of the Reactive algorithm but makes it compatible with hop-by-hop routing.

6.2.1 Algorithm

Effectively, evacuees only need the information that will get them to the location where they will receive their next path update. Indeed, the portion of their path beyond this point will be corrected and retransmitted by local nodes at each subsequent update, therefore there is no reason to transmit it ahead of time.
Our approach consists of issuing route updates every time an evacuee reaches a node – instead of doing so at fixed time intervals. This means we only need to inform the evacuees of the next node to reach: in practice, this is equivalent to hop-by-hop routing. Hence our method is to set the direction of each dynamic exit sign towards the first hop of the quickest path in the corresponding node's routing table.
As we have seen earlier, the Reactive algorithm is prone to oscillations. Associated to dynamic exit signs, routing algorithm oscillations could translate into frequent changes of direction being displayed. If these changes in direction occur frequently enough, there comes a point where some evacuees will see the sign pointing in multiple directions while they walk past it. This could either confuse them, or lead them to think the system is malfunctioning and should not be trusted. We prevent this by enforcing a "minimum display time" for each direction: while the routing algorithm runs continuously throughout the evacuation, the system only samples the quickest route and updates dynamic signs every Δt_R, the "update period". Figure 6.1 shows an example of directions that may be displayed by a dynamic sign, in a timeline format.

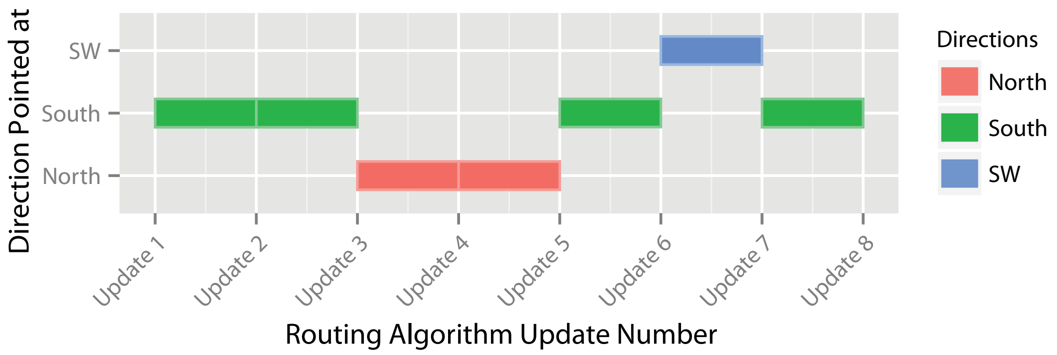


Figure 6.1: Timeline showing the directions pointed by a dynamic sign, driven using Reactive routing. Data created randomly for illustration purposes. The figure highlights that the sign can only display one direction per update cycle. However, the same direction can be displayed over several consecutive cycles.

6.2.2 Results

We implement the sign-driving algorithm introduced earlier and run simulations. We set Δt_R as an experimental variable with values: $\Delta t_R \in \{5, 10, 20, 40, 80, 120, 160\}$ seconds. We run a total of 840 simulations: 30 simulation runs for each combination of Δt_R value and evacuee headcount $|F| \in \{25, 50, 75, 100\}$ evacuees.

Sign Attendance and Update Period

We know that Δt_R corresponds to the minimum display time of a given direction. Setting it high enough makes the system more user friendly, as less evacuees are likely to see the sign changing direction while they walk past it. However, setting Δt_R excessively high will degrade the system's performance. To illustrate this, let us consider an extreme case where Δt_R is set to a value well above the building's evacuation time: the signs will never change from the initial direction during the evacuation and will effectively be *static*. Under these circumstances, no load balancing will take place as signs are unable to dynamically redirect evacuees. In fact, the evacuees will be routed according to the (sub-optimal) shortest-path policy, since this is the Reactive algorithm's initial solution, in the absence of any measured congestion (see 5.3.3).
Clearly, we need to find an optimal Δt_R value low enough to effectively redistribute the loads and high enough to improve evacuee compliance. Put differently, the problem is to set Δt_R as high as possible while allowing each sign to perform at least the essential r updates between the passage of the first and last evacuee. We refer to this period as the sign's "attended period"; and in contrast the "unattended period" covers any time

before the passage of the first evacuee, and after the departure of the last one. A sign's attended period will depend on its location: evacuees in "leaf" nodes (corresponding to offices, classrooms, etc.) will immediately leave these areas, and the attendance period of signs in these locations will be short. On the other hand, signs on the main egress paths and near the exits will be attended for longer. Finally, the signs' attendance time will also depend on the total evacuee headcount: the greater their number, the longer the evacuation, and the longer the sign's attendance periods.

We have demonstrated that, for our building graph, the critical flow-optimisation decision consists of distributing evacuees evenly on the staircases. Therefore, we will focus our analysis on this area of the graph. We have identified the set of signs which direct evacuees to one or the other staircase, which we will refer to as N_s. Figure 6.2 shows the cumulative percentage of evacuees which have walked past the signs on N_s at any point in time, from which we can derive their attendance period. The diagram *below* the main plot illustrates the update process of some higher values of $\Delta t_R \in \{20, 40, 80, 120, 160\}$: each alternation between yellow and blue corresponds to a new update. Both graphs are aligned along the time (horizontal) axis to count the number of updates that take place during a sign's attendance period.

Let us first consider the evacuation featuring the least amount of evacuees (25) on Figure 6.2. We can see that the bulk of the evacuees will have gone past the N_s signs before the first update for any $\Delta t_R \geqslant 120$ sec. This means that for any $\Delta t_R \geqslant 120$ and $|F| \leq 25$, any sign will point nearly all evacuees in the same direction: the signs will effectively be *static*. We also see that as the evacuee headcount increases so does the attendance time, and the signs will have time to perform more updates, for the same Δt_R value. For instance, when the evacuation features 100 evacuees, with any $\Delta t_R \leqslant 160$, the dynamic signs on N_s will have completed at least two update cycles before 90% of the evacuees have gone past. This gives the routing algorithm a better chance to divert some evacuees to balance the loads on both staircases.

Table 6.1 details, in terms of percentage, the maximum number of evacuees which walk past the signs on N_s, within one update period. For instance, if the sign's $\Delta t_R = 20sec.$ and $|F| = 50$, no more than 12% of the evacuees will be assigned to a staircase between two consecutive updates of the sign's display. This table effectively shows the sign's "resolution" in controlling groups of evacuees: as the percentages get smaller, the algorithm will have a finer control over smaller groups of evacuees, and operate at a higher resolution.

Based on this analysis, we can formulate two hypotheses on the system's behaviour, based on the Δt_R and $|F|$ values:

1. In theory, the system's performance improves as Δt_R reduces, as the system will be able to route smaller groups of evacuees. Yet we recall that setting Δt_R too low may limit the evacuees' compliance.

2. For identical Δt_R values, the system's performance will improve in evacuations with a larger headcount: a crowded building takes longer to evacuate, therefore the signs will have the opportunity to perform more update cycles where the balance of evacuees sent one each path can be more finely controlled.

	Evacuee headcount ($\|F\|$)			
Δt_R	25	50	75	100
5 sec.	6.9%	3.4%	2.3%	1.8%
10 sec.	12.6%	6.6%	4.5%	3.4%
20 sec.	22.9%	12.0%	8.1%	6.5%
40 sec.	36.5%	22.7%	15.9%	12.9%
80 sec.	67.3%	39.9%	31.3%	25.1%
120 sec.	90.3%	59.8%	44.1%	36.9%
160 sec.	95.9%	78.6%	57.7%	47.1%

Table 6.1: Maximum percentage of users which walk past the signs on N_s, between two consecutive sign updates. This can be related to the algorithm's "resolution": the smaller the group of evacuees passing by the sign between two consecutive updates, the higher the resolution. As the number of evacuee decreases and/or the update period increases this resolution lowers.

Flow Distribution

As we have demonstrated previously, the solution to the flow optimisation problem for our particular graph consists of dividing the flow of evacuees evenly across both staircases leading to the first floor. We know, from the previous Chapter, that the Reactive routing algorithm on its own, achieves this with a 5% error. We will now determine if the use of dynamic signs introduces an additional bias.

Figure 6.3 indicates the distribution of evacuees between the staircases: it shows the percentage of evacuees taking the central staircase – the remaining evacuees use the eastern staircase. The first hypothesis appears to be verified: overall, the system is more efficient at distributing evacuees when the values of Δt_R are smaller. We can see that the experiments with the lower Δt_R values have median staircase balance values within 0.05 of the optimal 0.5 (i.e. 1:1) ratio, and the standard deviation is also below 0.05, in most cases. As Δt_R increases, at first the system's performance does not seem

to vary, aside from a slight and gradual increase in the width of the distribution.
The pattern of relatively stable performance is suddenly broken past a Δt_R threshold value (shown on Table 6.2), which also depends on $|F|$. This sudden drop in performance seems to indicate the system has reached an operational limit and is no longer able to distribute the flow of evacuees evenly. This is characterised by a sudden increase in standard deviation ($\sigma \geq 0.15$), which indicates that the system's behaviour becomes less and less deterministic and predictable. The median staircase balance also increases by 15 to 20 percentage points, making it converge towards the shortest-path algorithm's results. This validates our initial assumption: as the directions are initialised according to the shortest-path policy, if the bulk of the evacuees go past the signs on N_s before the first update, the evacuee's trajectory are identical to shortest-path routing. We conclude that the performance degradation observed as Δt_R increases occurs because the signs do not change often enough during their attendance period.
We also notice that as the headcount increases, not only does the system perform better overall, but the threshold where the system's performance degrades is pushed into the higher values of Δt_R, both of which confirm the second hypothesis. For instance, in the evacuation featuring 100 evacuees, the system is able to distribute the evacuees evenly, all across the range of values assigned to Δt_R, with the first noticeable increase in median only occurring at the highest value of $\Delta t_R = 160$ sec.

Building Evacuation Times

Figure 6.4 shows the building evacuation times. The results of evacuations with low Δt_R values are comparable to the reference Reactive algorithm (i.e. without dynamic signs, examined in the previous chapter). Standard deviations are usually within the 30 seconds range, which is also comparable to those observed for the reference Reactive algorithm. We conclude that low Δt_R values allow the reactive sign-driving algorithm to accurately implement the routing algorithm's decisions. For higher Δt_R values, the sudden imbalance observed on Figure 6.3 past a certain stage, translates into a sharp increase in median evacuation times of up to 20% and a widening of the distribution with standard deviations reaching 100 seconds. This indicates that the bias introduced by signs is large and unpredictable. Table 6.2 shows the observed Δt_R threshold value where the system's performance starts to noticeably degrade[1].

We notice that regardless of the evacuee headcount, the system's performance degrades when approximately 60% or more evacuees have already been assigned to a staircase,

[1]We consider that the system's performance has degraded when the median evacuation times are 20% greater than those achieved with the original algorithm, i.e. without signs.

$\lvert F\rvert$	Δt_R Threshold	Max. users transiting within cycle
25	80 sec.	67.3%
50	120 sec.	59.8%
75	160 sec.	57.7%
100	>160 sec.	N/A

Table 6.2: Δt_R f steer the evacuees. The rightmost column shows the maximum number of users transiting past the signs on N_s between two consecutive updates (Tbl. 6.1).

before the signs perform their first update. We recall that the signs initially direct evacuees along the shortest path. This creates an imbalance in load, which the system tries to rectify from the first update onwards. If there aren't enough evacuees left to divert, so as to reestablish a balance in staircase load, there cannot be any improvement.
The explanation for the 60% threshold value can be found by comparing the shortest-path's staircase balance (80%-20%) with the target value (50%-50%). Let us consider an evacuation featuring 100 evacuees. If the 60 first evacuees have already been routed according to the initial shortest-path direction, there should be 12 on the eastern staircase and 48 on the central one. With 40 evacuees remaining, the system is still able to reestablish the 50%-50% balance of evacuees. In contrast, assuming 80 evacuees have already been routed before the first update, there would be 16 on the eastern staircase, and 64 assigned to the central staircase. By this time, even if the system directed the twenty remaining evacuees towards the eastern staircase, it would be unable to rectify the balance in staircase loads. In summary, there must be at least 40-30% evacuees left on the first floor before the signs perform their first update, otherwise it will be too late to rectify the imbalance.
Clearly, this limit will vary from a building topology to another, and will also depend on the distribution of evacuees. Yet this proves our general hypothesis that the Δt_R must be set to a small enough value so that the signs are able to update several times during their attendance period.

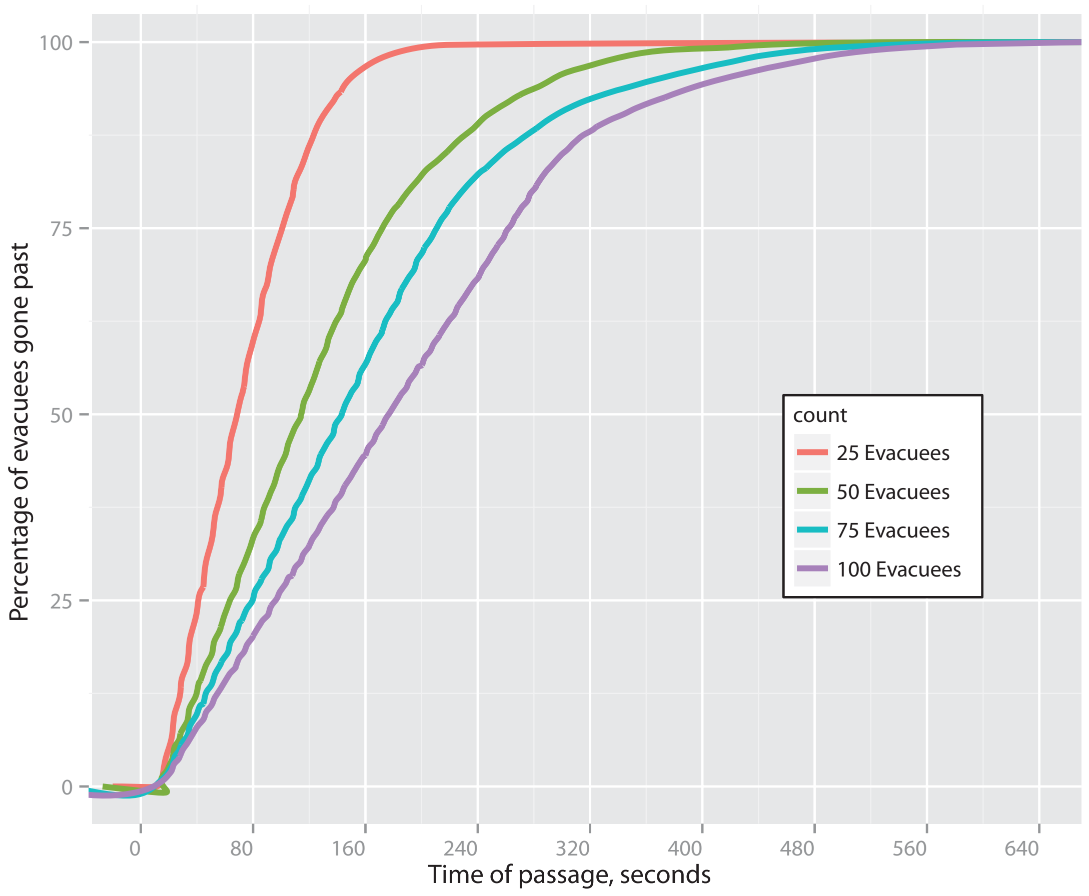
Route Analysis

Figure 6.5 shows the average evacuation path length. We notice that the paths shorten as the sign's update period increases. Once again, this is because the first directions displayed by the signs correspond to the shortest-path route. The later the first update occurs, the more evacuees will be routed through the shortest-path.
As we have seen in Chapter 5, the "standard" Reactive algorithm is liable to oscillations which result in evacuees needlessly walking back and forth between congested areas. The

results on Figure 6.5 indicate that the use of dynamic exit signs reduces this pattern without increasing evacuation times. This is because we have modelled the dynamic signs as being located just *before* the queuing areas. As the algorithm can only re-route *incoming* evacuees (and not those already in the queue) it shifts less evacuees from a staircase queue to another. This has a similar effect to the oscillation damping measure we have trialled (see 5.3.3), and in fact the results are comparable. Unlike the probabilistic route update approach, the sign-based algorithm does not require the re-routing probability parameter (p_r) set to a precise value to perform optimally, however Δt_R must be set low enough, as we have seen earlier.

6.2.3 Summary and Discussion

In this section we have converted the Reactive routing algorithm's output to hop-by-hop instructions which are compatible with the dynamic signs' display. To prevent signs from changing directions too often, we have enforced a minimum display time after each sign direction update, controlled by Δt_R, the sign update period. We have seen that our method is effective in directing evacuees so that building evacuation times are minimal. Our decision to place signs before the queuing area also limits the algorithm's propensity to shift evacuees from one staircase to another.
However, we have seen that Δt_R is an important performance factor. Low Δt_R values allow the signs to perform more updates during their attendance period; and as the groups of evacuees which walk past the signs between two consecutive updates are smaller, the algorithm has a finer control over the each evacuee's path, which is optimal for load balancing. We must also consider that updating signs too frequently may reduce the evacuee's confidence in the system, and therefore their compliance. As Δt_R increases, the performance is somewhat maintained up to the point where updates occur so rarely that most evacuees will have walked past a sign before it performs its first update. In this case the system will be almost static, and instead of performing load-balancing, it routes most evacuees along the shortest-path (as this is how the signs are initialised).
Another limitation of this approach is that signs are controlled independently of one another. While this is a advantage from a robustness point of view, it also means nothing ensures nearby signs point in consistent directions. This may in fact happen, since CPN is sub-optimal and has variable latency: nearby nodes may have slightly different views of the network and issue different recommendations. This could be addressed by implementing a distributed consensus algorithm [10] amongst nearby nodes.



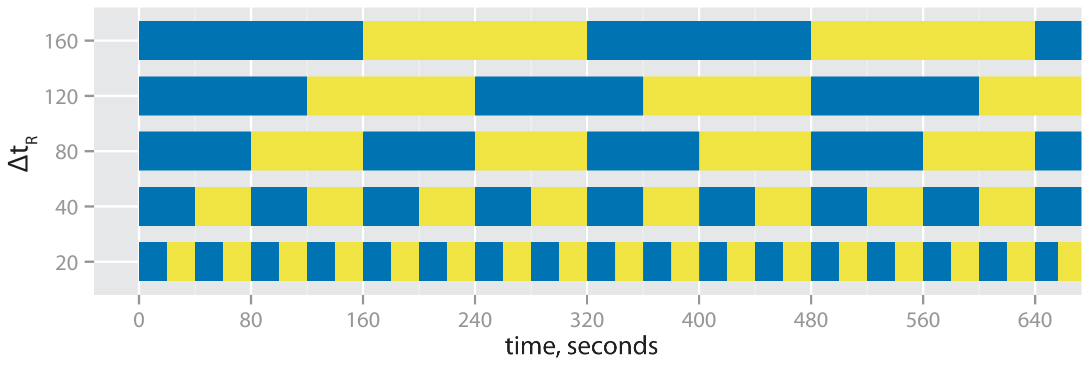


Figure 6.2: Cumulative percentage of evacuees which have passed in front of the signs on N_s. The timeline below illustrates the sign update process: each colour alternation corresponds to a new display update. Both graphs are aligned on the time axis to compare the update's chronology with the passage of evacuees. For instance, nearly all evacuees in the 25-evacuee scenario will have walked past the signs before their first update if $\Delta t_R = 160sec.$, whereas less than half of them will, when $|F| = 100$

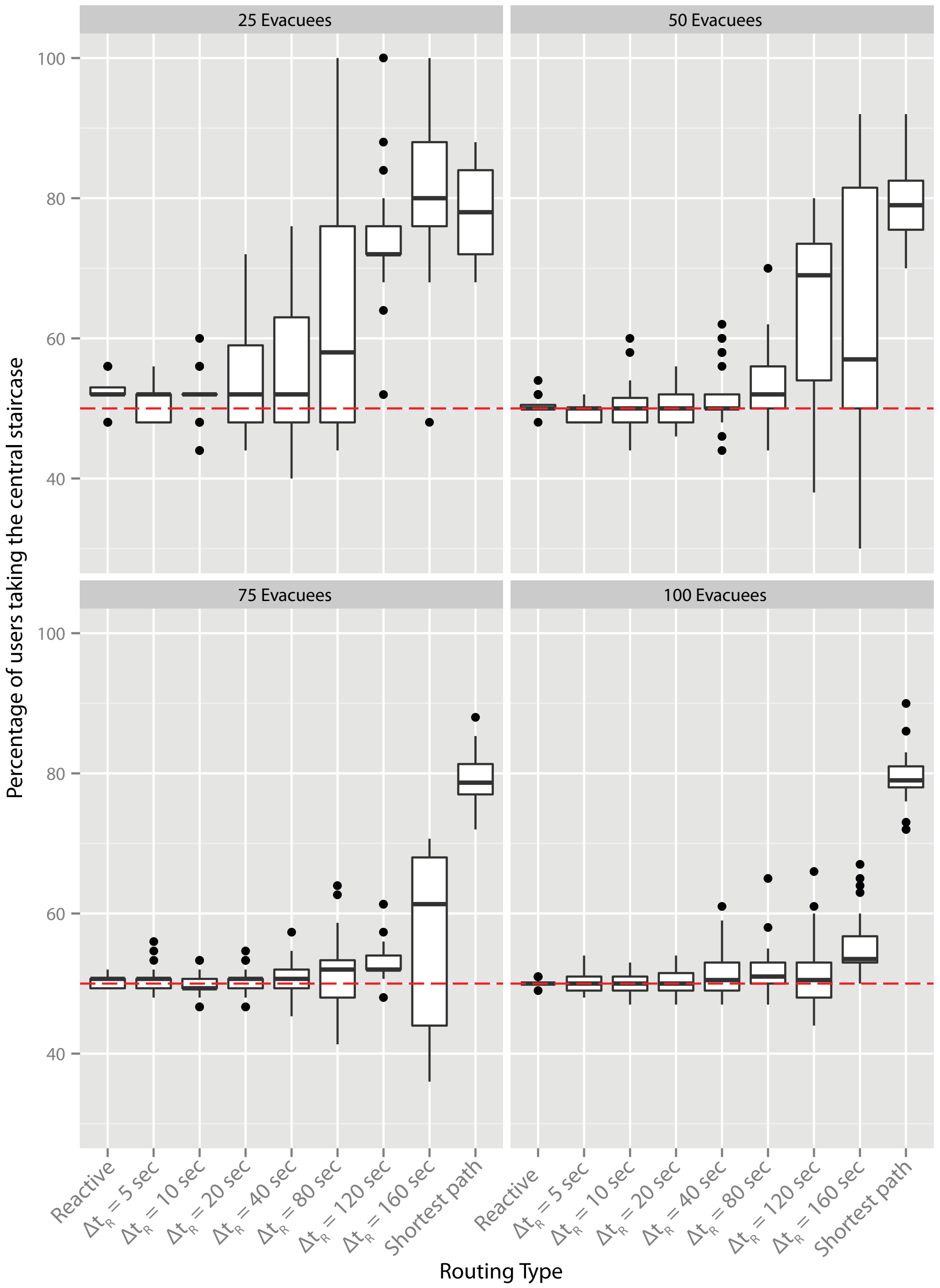


Figure 6.3: Ratio of users taking the central staircase to reach the first floor, Reactive routing with dynamic signs. The figure shows that the algorithm's ability to balance loads on the staircases decreases with smaller evacuee counts and/or longer update periods.

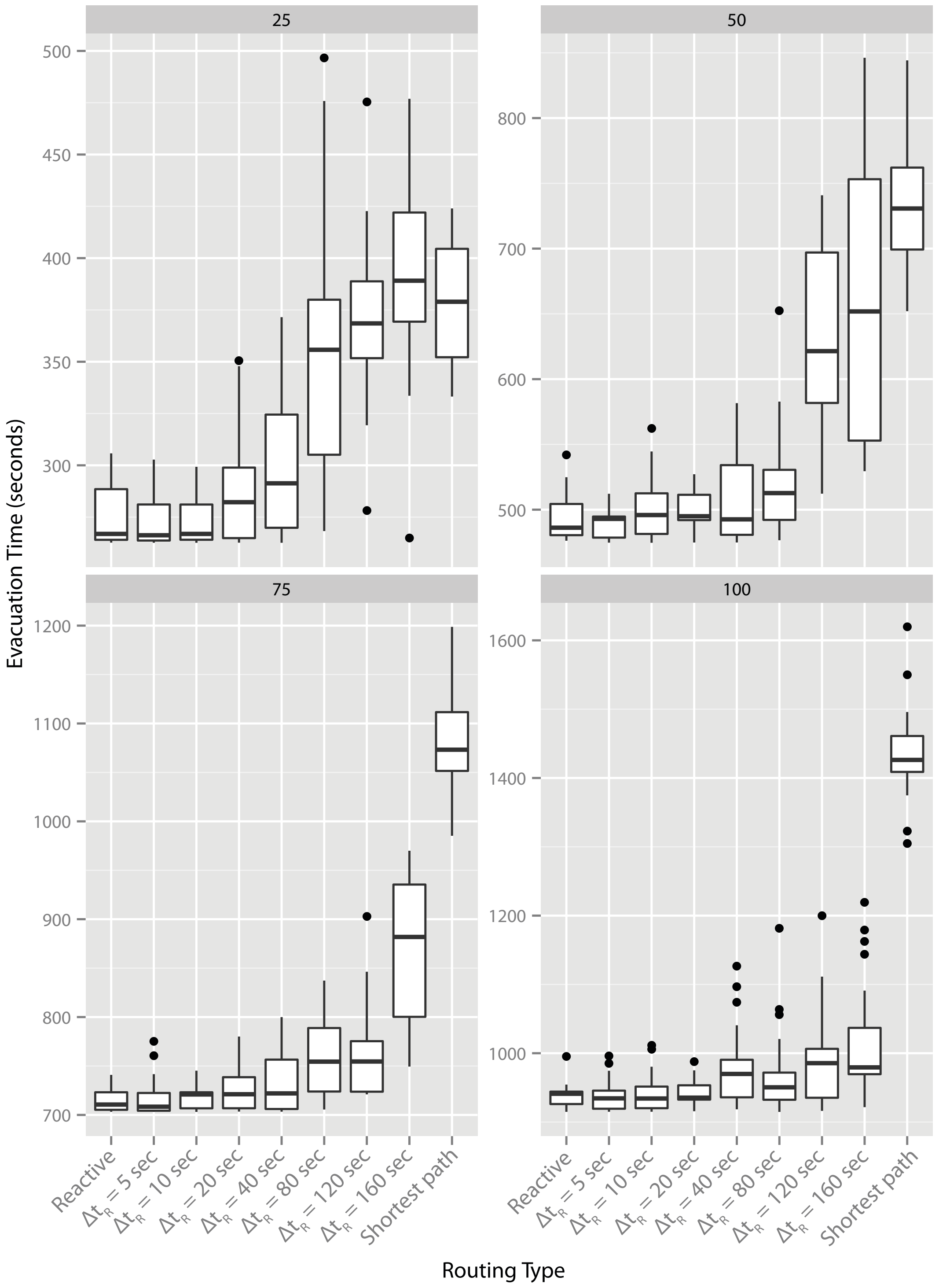


Figure 6.4: Building evacuation times, Reactive routing with dynamic signs.The figure shows that past a certain Δt_R threshold, the evacuation times suddenly increase, and converge towards the values of the shortest path approach.

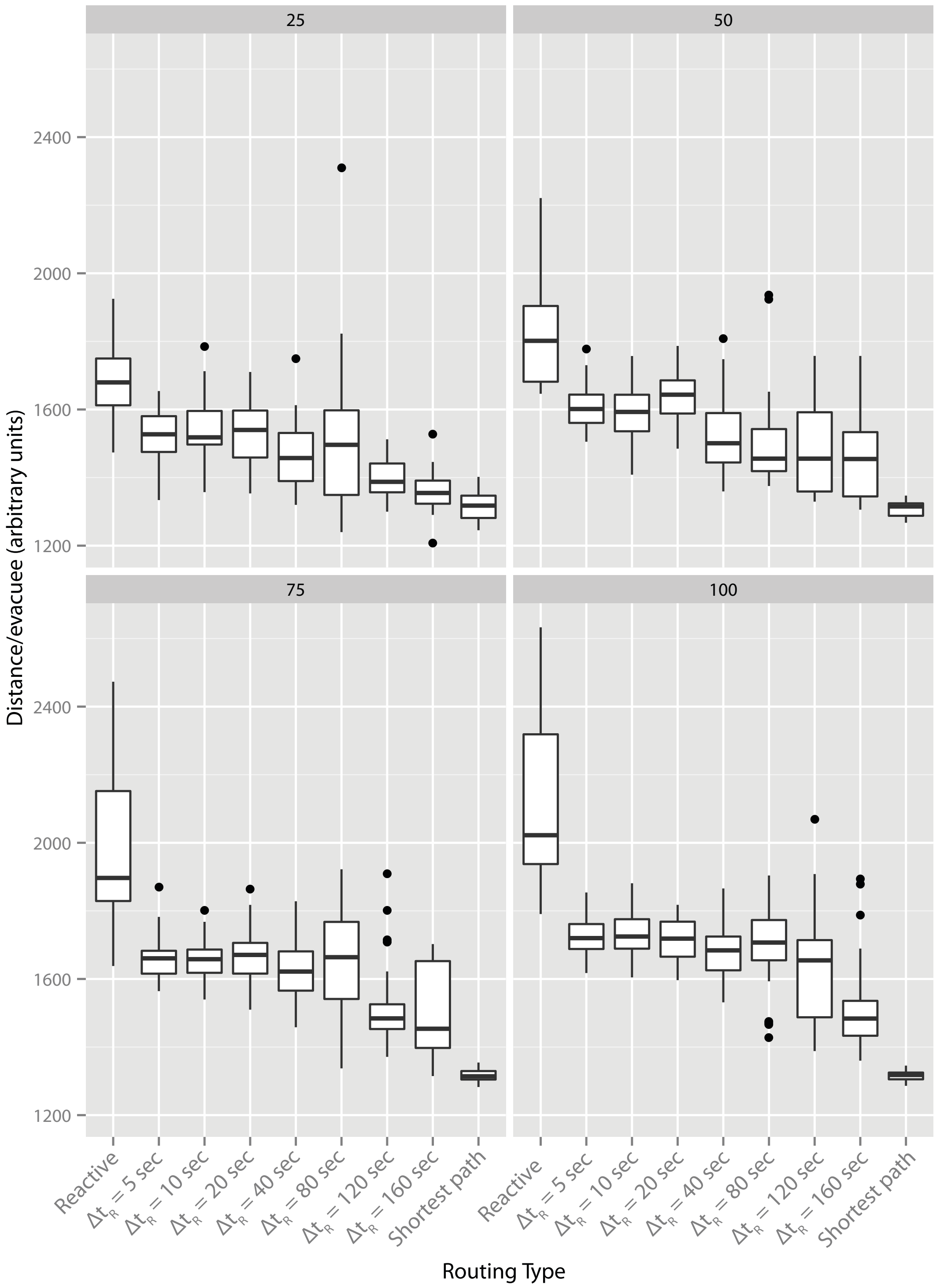


Figure 6.5: Average distance walked by evacuees, Reactive algorithm with dynamic signs. As Δt_R, the evacuee path length decreases as the system's behaviour approaches shortest-path routing.

6.3 Proactive Routing Using Signs

The Proactive routing algorithm performs source-routing, and the routes are *definitive* (i.e. never corrected or updated) and assigned *individually* to each evacuee. This is incompatible with the hop-by-hop approach mandated by the dynamic exit signs we wish to use. In this section we present an algorithm which addresses this problem without adding any sensory requirements beyond those of the original Proactive algorithm.

6.3.1 Approach

The most straightforward approach consists of breaking down each evacuee's route into next-hop instructions, and transmitting these instructions to the relevant dynamic signs. The dynamic signs would then identify each evacuee as he approaches the sign, and display the corresponding next-hop direction as he walks past.

Formally, as the evacuee f walks past the sign on node n, the sign points towards whichever direction e verifies $\pi_{n\to e}(f) = 1$. The $\pi_{n\to e}(f)$ correspond to next-hop routing probabilities:

$$\pi_{n\to e}(f) = \begin{cases} 1, & \text{if } e \ \in P_f \text{ and } f \text{ is at node } n \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

where P_f is the path assigned to f by the Proactive routing algorithm.

Identification Requirement Removal

This simplistic implementation is impractical for many reasons, one of them being that the system must *uniquely* identify evacuees. This additional sensory requirement goes against our objective to limit the reliance on sensors during the evacuation. Therefore, we aim to design an algorithm which produces next-hop routing probabilities that are independent of f.

Our model assumes there is only one class of users, that is, the differences between evacuees are small enough that we can consider them as all having the same characteristics (walking speed, abilities, etc.). We do not consider a multi-class system at this stage, as it would require as many signs as there are classes of users: for instance one sign for the able-bodied, one for the disabled, one for the elderly, and so on. Thus from an evacuee's point of view, we consider that exchanging paths with another evacuee would make no difference – as long as both paths begin at the evacuees' current location. From the system's point of view, if two users with the same mobility model and parameters cross each other's path at the same time and happen to exchange their respective paths, they would travel on their “new” path at the same speed and in the same way as the original path owner would have, and it ensues that the congestion in the building would remain unchanged. This means the flow optimisation will not change if path are exchanged between users which are at the same location at the same time. This principle allows us to “relax” the routing probabilities so that they are no longer evacuee-dependant but only time- and location-dependant:

$$\pi_{n\to e}(t) = \frac{|F_{n\to e,t}|}{|F_{n,t}|} \tag{6.2}$$

where $F_{n,t}$ represents the set of of evacuees departing node n at instant t, and of which $F_{n\to e,t}$ depart via edge e. This formula means that next-hop directions can be redistributed arbitrarily amongst evacuees located at the node at the same time.
Equation 6.2 is valid in continuous-time and only concerns evacuees which arrive at the *same time*. However our system’s maximal temporal resolution is Δt_S: the time span covered by a time-bin, therefore we must expand Formula 6.2’s applicability to discrete-time. Let us redefine $\pi_{n\to e}(t)$ using its arithmetic mean $\overline{\pi}_{n\to e}(k)$ over the span of the kth time-bin, and a discrete deviation factor $\varepsilon_\pi(t)$:

$$\pi_{n\to e}(t) = \overline{\pi}_{n\to e}(k) + \varepsilon_\pi(t) \tag{6.3}$$

where $\overline{\pi}_{n\to e}(k)$ is obtained by analysing the set of paths issued by the Proactive routing algorithm. In particular, we summarise the next-hop directions taken from node n over the duration of the k^{th} time-step:

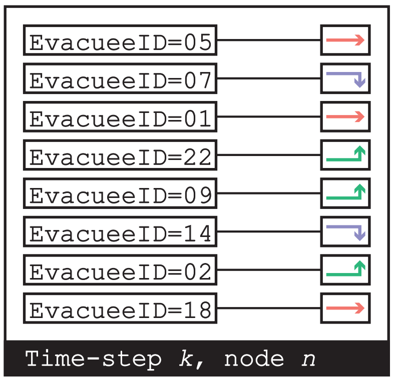
$$\overline{\pi}_{n\to e}(k) = \frac{|F_{n\to e,k}|}{|F_{n,k}|} \tag{6.4}$$

Let us assume[2] $\pi_{n\to e}(t)$ can be considered invariant over the span of any time-bin k, which means $\varepsilon_\pi(t)$ can be considered null and we can replace $\pi_{n\to e}(t)$ by $\overline{\pi}_{n\to e}(k)$.
This means next-hop directions can be exchanged amongst evacuees which reach the same node within the same time-step, but the tradeoff is that the routing probabilities $\pi_{n\to e}$ are now fixed over the span of a time-step. The sign-driving algorithm is now free to arbitrarily re-map the relationship between next-hop directions and evacuees, as long as the bijective nature of the map is preserved.

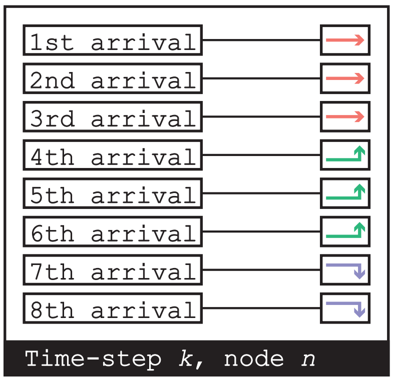
[2]For the sake of clarity, all assumptions made in this paragraph will be reviewed and analysed separately in a following section.

Stabilising the Sign Display

The algorithm we have conceived to this point forwards evacuees one-by-one without having to identify them. Yet the assumption that evacuees can be forwarded on a one-by-one basis is still unrealistic. For instance, if a compact group of evacuees walked past a sign, there would be no way to send each evacuee in separate directions. The evacuees would see the sign switching directions at a high rate and would only be confused. To "stabilise" the display, we make the sign point in a direction only once per time-step and for as long as possible. This is done by sorting and grouping all next-hop instructions and assigning them to evacuees based on their order of arrival. Figure 6.6 illustrates this process: Fig. 6.6a shows the raw information extracted from the collection of routes assigned to evacuees by the Proactive routing algorithm, where each next-hop direction is associated to a particular evacuee ID. Figure 6.6b shows the completed re-allocation process: the next-hop directions have been disassociated from their original evacuee ID and ordered and grouped. They are no longer assigned based on evacuee ID, but based on the evacuees' order of arrival. The system no longer needs to uniquely identify evacuees, but must still be able to determine how many evacuees have walked past the sign, in order to determine when to switch to the next direction. Figure 6.6b also confirms the sign will only display each direction once during the entire time-step.



(a) Raw information extracted from the evacuee's routes.



(b) Directions ordered and re-assigned based on evacuee's order of arrival.

Figure 6.6: Figure showing the next-hop direction of each evacuee, as assigned by the Proactive algorithm; and how our algorithm re-assigns these directions (grouping and ordering) to stabilise the dynamic's sign display.

Presence Detection Requirement Removal

As we have mentioned before, our algorithm still relies on presence-detection sensors to count the evacuees that have passed by a sign. We can remove this constraint by taking a flow-based approach: instead of counting users, let us use $\lambda_n(t)$, the instantaneous flow of evacuees walking past the sign. We wish to determine $\delta t_{n \to e,k}$, the duration for which the sign should point towards e within the k^{th} time-step. In order to direct the prescribed number of evacuees $F_{n \to e,k}$ in that direction, we must set $\delta t_{n \to e,k}$ so that:

$$\int_{\delta t_{n \to e,k}} \lambda_n(t)\, dt = |F_{n \to e,k}| \tag{6.5}$$

Measuring $\lambda_n(t)$ is likely to require some form of presence detection sensors, but its arithmetic mean over the k^{th} time-step $\overline{\lambda_n}(k)$ can be derived from the number of reservations made in the corresponding time-bin:

$$\overline{\lambda_n}(k) = \frac{|F_{n,k}|}{\Delta t_S} \tag{6.6}$$

We can now substitute $\lambda_n(t)$ with its arithmetic mean and a discrete error component $\varepsilon_{\lambda_n}(t)$. This lets us re-write equation 6.5:

$$\int_{\delta t_{n \to e,k}} \left[\, \overline{\lambda_n}(k) \; + \; \varepsilon_{\lambda_n}(t)\, dt\right] = |F_{n \to e,k}| \tag{6.7}$$

We can simplify this equation by assuming the interval between two consecutive arrivals is a random variable distributed evenly and narrowly around a mean value. In this context, the consecutive values of $\varepsilon_{\lambda_n}(t)$ are distributed around zero, and an integral of $\varepsilon_{\lambda_n}(t)$ tends towards zero. Under this assumption, equation 6.7 becomes:

$$\int_{\delta t_{n \to e,k}} \overline{\lambda_n}(k)\, dt = |F_{n \to e,k}| \tag{6.8}$$

We are left with integrating a constant to obtain $\delta t_{n \to e,k}$:

$$\delta t_{n \to e,k} = \frac{1}{\overline{\lambda_n}(k)} \cdot |F_{n \to e,k}| \tag{6.9}$$

We reformulate Equation 6.9 using the definition of $\overline{\lambda_n}(k)$ (Eq. 6.6), and see that the duration for which a sign points in a direction ($\delta t_{n \to e,k}$) is proportional to the number of evacuees which should be sent this way ($|F_{n \to e,k}|$), which is the mean routing probability

($\overline{\pi}_{n\to e}(k)$) defined in Equation 6.4:

$$\begin{aligned}\delta t_{n\to e,k} &= \Delta t_S \cdot \frac{|F_{n\to e,k}|}{|F_{n,k}|} && (6.10)\\ &= \Delta t_S \cdot \overline{\pi}_{n\to e}(k) && (6.11)\end{aligned}$$

Summary

In summary, our approach first consists of aggregating all routes assigned by the Proactive algorithm to determine the ratios of routes following a particular outbound direction. Then, each sign is scheduled to cycle through every outbound direction each time-step, where each direction is displayed for a duration which is proportional to the number of evacuees which should be sent this way.
This sign-driving algorithm is at the same time simple to implement and does not require any sensory input – beyond what is required for the original Proactive routing algorithm. However, the algorithm we have described operates the following assumptions:

1. The Proactive algorithm requires a representative mobility model. This is required for capacity reservations to be accurate.
2. The system also requires an accurate measurement of the evacuee's location at the time when the evacuation begins. This lets the Proactive routing algorithm create a flow-optimal set of paths for each evacuee.
3. Evacuees must walk at a reasonably similar pace. This allows us to exchange paths under certain conditions and dispenses us from individually recognising evacuees.
4. The instantaneous rate of arrival in front of a sign varies randomly and is centred around a mean value. This lets us replace the instantaneous flow value by its arithmetic mean and dispenses us from physically counting the passage of evacuees in front of the sign.
5. The probability of routing evacuees in any direction can be considered invariant over a time-step. This allows us to reassign next-hop directions arbitrarily within the time-step, and in particular group directions together.

We will explore the limits imposed by each assumption after providing a formal definition of the algorithm.

```
Data: buildingMap;timeBins;userDensityMap
Result: Schedule of directions for each sign

/* Path Allocation - capacity reservation                              */
1  //refer to Algorithm 1

/* Resolve Flows for each outgoing edge                                */
2  forall the node in buildingMap do
3      forall the k time-steps in K do
4          totalVisits ← $|F_{node,k}|$;
5          if totalVisits = 0 then
6              ratios(node,k) ← null; //prevent div.  zero
7          else
8              forall the outEdge in node.getOutgoingEdges() do
9                  nextHopCount(outEdge) ← $|F_{node \to outEdge,k}|$;
10                 ratios(node, k, outEdge) ← nextHopCount(outEdge)/totalVisits
11             end
12         end
13     end
14 end
/* At this stage, ratio(node,k,outEdge) contains the ratio of
   evacuees going from node towards outEdge in the $k^{th}$ time-bin */

/* Calculate durations within sign cycle                               */
15 forall the node in buildingMap do
16     forall the k time-steps in K do
17         startTime ← k $\times \Delta t_S$;
18         if (ratios(node,k) = null) then
19             //Cannot compute sign direction, default to shrt.  path
20             endTime ← $(k+1) \times \Delta T_S$
21             signSchedule(node, startTime, endTime) ← getShrtPath(node);
22         else
23             forall the nextEdge in shuffle(node.getOutgoingEdges()) do
24                 duration ← $\Delta t_S \times$ ratios(node, k, nextEdge);
25                 endTime ← duration + startTime;
26                 signSchedule(node, startTime, endTime) ← nextEdge;
27                 startTime ← endTime; //prepare for next iteration
28             end
29         end
30     end
31 end
32 return signSchedule
```

Algorithm 2: Flow computation and sign setup algorithm.

6.3.2 Algorithm

An outline of the path-assignment procedure is provided in Algorithm 2. The algorithm is divided into three sections: *1)* route assignment, *2)* flow aggregation and *3)* sign setup.

1. **Route assignment** Routes are assigned following the randomised variant of the Proactive algorithm. Refer to Algorithm 1 in 5.2.2 or the flowchart on Figure 5.2.

2. **Flow aggregation** The algorithm iterates over all nodes and time-steps (lines 2 and 3), and if there are capacity reservations made during this time-step, it counts the number of evacuees continuing onto every outgoing edge (ln.9) and converts it into a ratio (ln. 10) by dividing the total number of expected visits, calculated on line 4. In practice, this ratio is the routing probability $\overline{\pi}_{n\to e}(k)$ from Equation 6.4. At the end of this step, the algorithm will have determined the ratios of users going through each onward node for every node and every time-step.

3. **Sign setup** If there are no evacuees expected to transit through a node at a given time-step (unattended period), we set the dynamic sign to a default value corresponding to the shortest-path (ln. 21). This default value is required as the algorithm cannot calculate outbound direction ratios if there are no capacity reservations made in the time-bin (see ln. 6). Otherwise, the dynamic signs' schedule is set so that the time a sign points towards a given direction ($\delta t_{n\to e,k}$) is proportional to the ratio of evacuees taking this outbound direction and the duration of a time-step Δt_S (ln. 24). The algorithm iterates through every node and every time-step (lns. 15 and 16) and schedules the display times of each outbound direction one after the other (ln. 25). The order in which which directions are displayed is randomised (ln. 23) so that the sequence order varies from a time-step to another. Shuffling the sequence order in which directions are pointed by the sign helps to reduce the algorithm's bias in certain conditions, as we will see later.

The output of the algorithm is a schedule of directions to point at, for each dynamic exit sign. A sample output of the algorithm is presented in Figure 6.7.

6.3.3 Review of Assumptions and Limitations

The algorithm we have presented adds several assumptions beyond those already made to implement Proactive routing. We will now review these assumptions, assess their validity and envisage the limits they may impose on the performance of the evacuee guidance system.

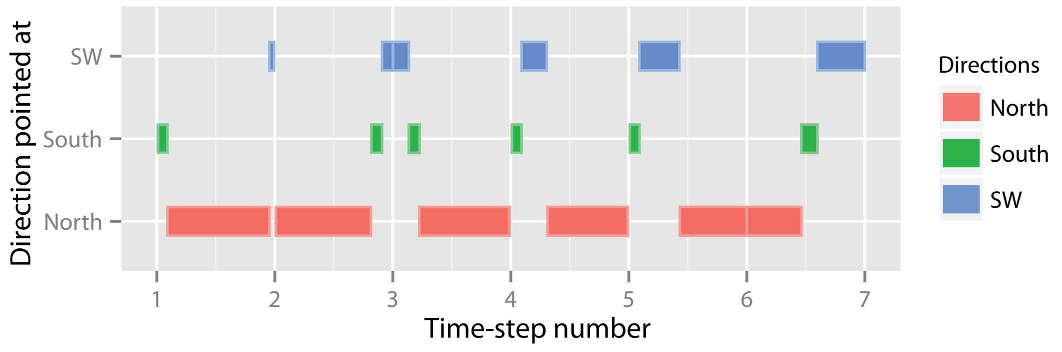


Figure 6.7: Output of the Proactive sign-driving algorithm: a schedule of directions for the dynamic sign. The sign displays all directions with non-null routing probabilities in each time-step. We can also see that the order in which directions are displayed inside each time-step is randomised. Diagram for illustrative purposes based on randomly-generated data.

Invariance of Routing Probability Over a Time-Step

In most evacuation problems, the routing probabilities will be time-variant: generally, the first few evacuees are routed along the shortest-path, up to the point where it becomes saturated; at this stage, the routing probabilities towards diversions start to increase. Routing probabilities are also expected to change in the same manner with predicted surges or decreases in arrival rates.

To re-assign routes to evacuees arbitrarily within a time-step interval, we have assumed that the routing probabilities $\pi_{n \to e}(t)$ were invariant over the span of a time-step. In the context of an evacuation, this assumption is likely to be valid if the duration of a time-step is in the same order of magnitude as the transit time of a few evacuees across a typical edge. The smaller the time-step the more often the algorithm will be able to update the routing probabilities, and the finer the control over the evacuee's path. However – much like the Reactive algorithm – by reducing the time-step duration, each direction will be displayed for a proportionally shorter duration and the sign's display will appear less stable from an evacuee's perspective. As we have explained, this is likely to lower the evacuees' compliance with the system.

Routing probabilities are also more likely to remain stable if a steady-state regime appears where arrival rates become steady. This is likely to occur where evacuees form homogeneous groups departing from nearby locations, e.g. stadiums, theatres, etc. On the other hand, if evacuees are scattered across the building, a steady-state is less likely to occur. Thus, we expect our algorithm to perform better for a larger evacuee headcount where evacuee densities are higher.

Grouping of Directions

To reduce confusion amongst evacuees, the algorithm can only display a given direction once per time-step. While this reduces the likelihood of "breaking up" groups of evacuees travelling together, this also means that the flow on the downstream edges will form a "square wave" with a cycle time corresponding to the time-step, as illustrated by Figure 6.8.

As a result, evacuees arrive to the next node by "waves" and their flow is autocorrelated on the scale of a time-step. This is undesirable, since our algorithm assumes inbound flows vary randomly over a time-step; and modifying the time-step duration will have no effect since all nodes apply the same time-step. However, randomising the order in which directions are displayed at each time-step can help reduce the systematic bias this phenomenon may cause: we will characterise this bias in the next paragraph and explain how randomisation can help.

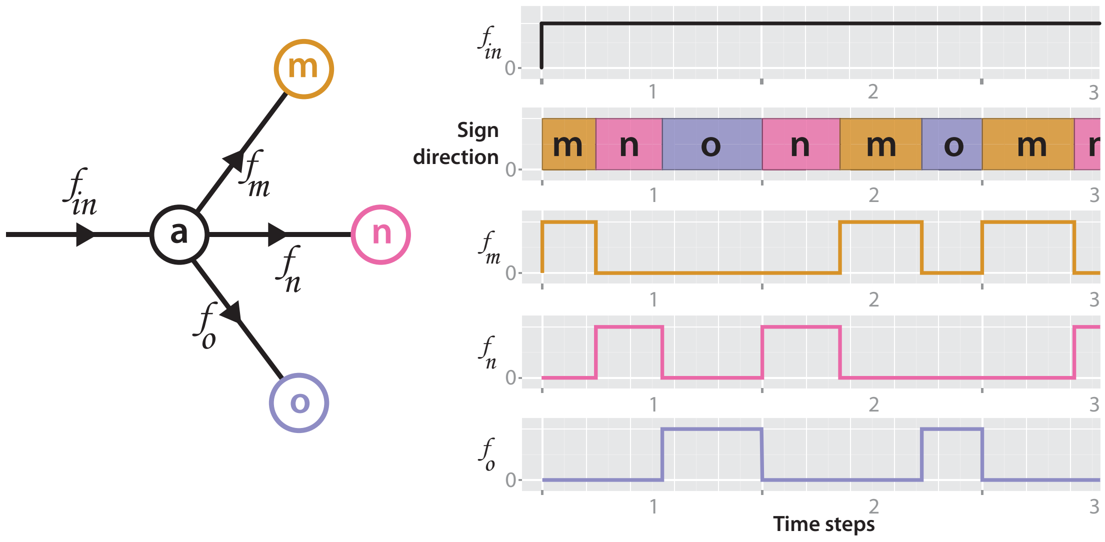


Figure 6.8: Graphical representation of how the sign sends waves of traffic down each outbound link. The inbound flow into a is steady, but is broken up into discrete groups as the sign directs groups towards each outbound edge.

Random Variations in Incoming Flow

We have seen in the previous paragraph that grouping directions displayed by the sign makes evacuees arrive by waves. Yet our system assumes evacuees arrive at a somewhat steady rate of $\overline{\lambda}_n(k)$ over a time-step. Let us first consider the bias if the system did not group directions, i.e. if each incoming evacuee could be individually assigned a next-hop direction at random, according to the routing probabilities. The number of evacuees routed in direction e during the k$^{\text{th}}$ time-bin is:

$$|F_{n\to e,k}| = \pi_{n\to e}(t) \cdot \left[\int_{\Delta t_S} \overline{\lambda_n}(k)\ dt + \int_{\Delta t_S} \varepsilon_{\lambda_n}(t)\ dt\right] \tag{6.12}$$

By definition, the integral of a function's deviation to its mean over the averaging interval equals zero. The second integral disappears, and we prove that the number of evacuees routed in a direction is proportional to the routing probability, as intended by the system. This system is not influenced by variations in the incoming rate of evacuees. However, when we group directions together, directions are no longer assigned randomly, but on a time-dependent basis:

$$\pi_{n\to e}(t) = \begin{cases} 1, & t \in \delta t_{n\to e,k} \text{ (i.e. while the sign point towards } e\text{)} \\ 0, & \text{otherwise} \end{cases} \tag{6.13}$$

Therefore we can reduce the span of the integral from the entire time-step Δt_S to just $t \in \delta t_{n\to e,k}$: since $\pi_{n\to e}(t) = 0$ for $t \notin \delta t_{n\to e,k}$ the integrals are multiplied by a zero-probability and disappear.

$$\left|F'_{n\to e,k}\right| = \int_{\delta t_{n\to e,k}} \overline{\lambda_n}(k)\ dt + \int_{\delta t_{n\to e,k}} \varepsilon_{\lambda_n}(t)\ dt \tag{6.14}$$

Looking at this equation, we recognise the first integral as $|F_{n\to e,k}|$ (Eq. 6.8) the number of evacuees we *intend* to guide. We define the second integral as $\varepsilon_{n\to e,k} = \left|F'_{n\to e,k}\right| - |F_{n\to e,k}|$: the number of evacuees *unintentionally* guided in direction e because of biased variations of $\lambda_n(t)$ during the time-step.
We acknowledge that $\varepsilon_{n\to e,k}$ and $\left|F'_{n\to e,k}\right|$ must be integers, yet our calculations are based on a continuous flow $\lambda_n(t)$ and may produce real numbers. To address this we would also need to incorporate a "rounding" error component taking values between -0.5 and +0.5, to reflect the reality that we are routing indivisible entities. For the sake of simplicity, we consider this rounding error negligible.
As $\varepsilon_{\lambda_n}(t)$ may vary arbitrarily during the time-step, the $\varepsilon_{n\to e,k}$ will depend on the values taken by $\varepsilon_{\lambda_n}(t)$ while the direction is being displayed by the sign. If $\varepsilon_{\lambda_n}(t)$ varies randomly around zero over the display period of a particular direction, its integral will tend towards zero, and $\varepsilon_{n\to e,k}$ will also tend towards zero, as shown on Figure 6.9, 2nd time-step. However if this is not the case, there will be a bias towards some directions.

- If $\varepsilon_{\lambda_n}(t)$ is predominantly positive during the direction's display period, $\varepsilon_{n\to e,k} > 0 \Rightarrow \left|F'_{n\to e,k}\right| > |F_{n\to e,k}|$. The number of evacuees guided in this direction will be

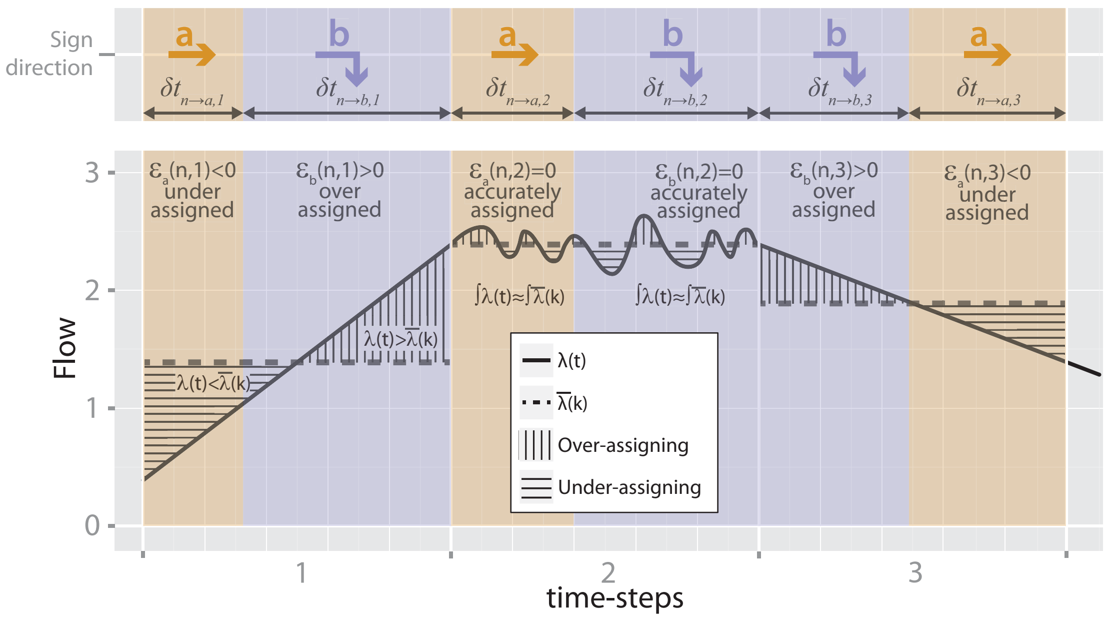


Figure 6.9: Graphical illustration of the effect biased $\lambda_n(t)$ values have on the system. In the 2nd time-step, the arrival rate varies randomly. On the first and third time-steps, the arrival rate is autocorrelated (linearly increasing or decreasing), and as a result, the direction is either over- or under-assigned. The hatched areas correspond to either $\varepsilon_{\lambda_n(t)} > 0$ (vertical hatching) and $\varepsilon_{\lambda_n(t)} < 0$ (horizontal hatching).

above the target number by $\varepsilon_{n\to e,k}$. This occurs when there is a surge of evacuees while the direction is being displayed. An example of this is appears in Figure 6.9, while the sign displays direction b in the 3rd time-step.

- If $\varepsilon_{\lambda_n}(t)$ is predominantly negative during the direction's display period, $\varepsilon_{n\to e,k} < 0 \Rightarrow \left|F'_{n\to e,k}\right| < |F_{n\to e,k}|$. The number of evacuees guided in that direction will be below target, by $\varepsilon_{n\to e,k}$. This occurs when the arrival rate is below average while the direction is being displayed. An example of this is appears in Figure 6.9, while the sign displays direction a in the 3rd time-step.

Figure 6.10 is a schematic example of what happens when evacuees arrive as a group over the span of multiple time-steps. In the second time-step, evacuees arrive at a constant rate and the system directs them as intended, while in the first time-step they only start arriving towards the end and only see the last direction pointed by the sign. The same applies to the third time-step where evacuees arrive early and only see the first direction displayed.

We have defined $\varepsilon_{n\to e,k}$ as the number of evacuees unintentionally routed over the

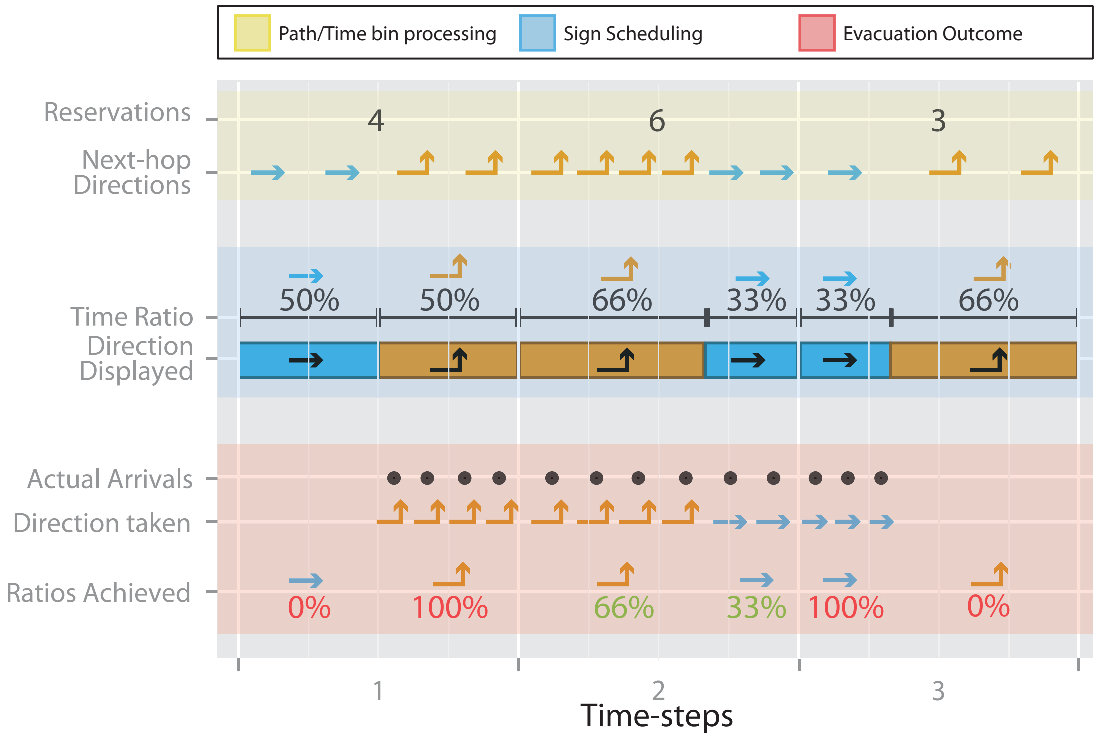


Figure 6.10: Graphical representation of the impact of correlated evacuee arrivals. The top line shows the information extracted from the routes. The middle section shows the computations carried out by the sign-driving algorithm. The bottom section shows a possible outcome, where evacuees arrive in a dense group over the span of multiple time-steps. As evacuees do not arrive at a steady rate during over first and third time-step, some directions are over-/under-assigned.

display period of *one* direction ($\delta t_{n \to e,k}$). We now want to find the total number of *unique* evacuees which were not routed according to plan because $\varepsilon_{\lambda_n}(t) \neq 0$. Due to the basic flow-conservation constraints, evacuees that were "missed" from one direction inevitably become "extra" evacuees for another direction. Thus each mis-routed individual is counted *twice*: once as a negative error, for the *intended* direction he was *not* sent to, and once more as a positive error for the direction he ends up being *erroneously* routed to. The signed sum of errors will therefore be zero:

$$\sum_{E_n} \varepsilon_{n \to e,k} = 0 \tag{6.15}$$

Where E_n corresponds to all outbound directions for the node n. The number of instances of evacuees being erroneously routed at a node, for the entire evacuation is:

$$Z = \frac{1}{2} \sum_{K} \sum_{N} \sum_{E_n} |\varepsilon_{n \to e,k}| \tag{6.16}$$

Our limited knowledge of the function $\lambda_n(t)$ makes it impossible to minimise Z analytically. However we can consider that shorter time-steps will leave less room for $\lambda_n(t)$ to deviate significantly from its mean, and thus we expect to see lower errors as the time-steps shorten. Put differently, we expect that the mean values will better "track" the instantaneous values if we break the function into smaller time windows. Again, we have come to the conclusion that low Δt_S values theoretically improve the system's performance, however, set too small, the evacuee compliance will likely be reduced.

Randomisation of Directions

Randomising the order in which directions are displayed plays a role in limiting the appearance of systematic bias towards a particular direction. To illustrate this, let us consider a case where $\lambda_n(t)$ is linearly increasing over the span of several time-steps. The arrival rate will always be below average at the beginning of every time-step, and the first direction displayed will be "short" of some evacuees. Conversely, the arrival rate will become greater than its mean towards the end of the time-step and the last direction displayed will have "extra" evacuees. If directions are displayed in the same order, these errors will accumulate over the time-steps and a systematic bias will occur in the assignment of some directions. On the other hand, if we randomise the order of directions, the errors are expected to compensate each other over a few time-steps.
This will also be effective if the arrival rate is a periodic function. We have mentioned that we expect evacuees to arrive as waves, and by randomising the order in which directions are displayed, these waves will arrive at a different moment in each time-step, as shown on Figure 6.8. This means the over-assignment which occurs when the wave arrives will not always affect the same direction. While this will not remove the error caused by "wave arrivals", it is likely to prevent systematic bias.

Unplanned Movement of Evacuees

Both Proactive routing algorithm and associated sign-driving algorithm rely on an evacuee mobility model to forecast the evacuee's location, which in turn is used to optimise the evacuation process. This mobility model makes some strong assumptions: evacuees can be regarded as identical from the model's point of view, and they have the same

walking speed.
We have explained in section 3.3 that the variance in evacuee walking speed decreases as congestion increases. As most edges are saturated during evacuation, this makes our assumption of uniform walking speed relevant. However congestion may take some time to form, and the first evacuees will experience free-flow conditions where differences in walking speed may be non-negligible. These evacuees will arrive earlier than forecast, which is why we set the signs to point towards the shortest-path by default.
The performance of this system is likely to be most affected if the evacuee's mean speed is under- or overestimated, as the sign's display will not correspond to the expected arrival of evacuees, and the flow-optimal solution is also likely to be less valid. Therefore precisely estimating each edge's transit delay is a critical requirement for the system's performance.
The algorithm also operates under the assumption that the evacuees do not move while the routes are being computed and the signs set up. If the routing and sign-driving algorithms are too slow, the evacuees may decide to start moving without waiting for the system to provide guidance and will arrive ahead of the forecast time. It is therefore critical to ensure sufficient computing power to allow all algorithms to execute within a few seconds at most.

6.3.4 Results

We have seen in the previous section that the sign-driving algorithm assumes invariance of routing probabilities and random flow variations over a time-step. We have postulated that these assumptions are more likely to be valid if time-steps are small, and if the evacuee count is higher.
In this section we will attempt to validate these hypotheses by exploring simulation results with $\Delta t_S \in \{0"18, 1"12, 2"24, 3"36, 4"48\}$. We run a total of 600 simulations: 30 randomised simulation runs for each combination of Δt_S value and evacuee headcount $|F| = \{25, 50, 75, 100\}$ evacuees.

Error Introduced by the Signs

As we have seen in Chapter 5, the Proactive algorithm is run only once at the beginning of the evacuation and assigns an individual and definitive route to each evacuee. Then, the sign-driving algorithm (Algo. 2) post-processes the output of the routing algorithm to determine a schedule of directions for each sign. This two-phase operation lets us dissociate the process and measure the error introduced at each step. We setup the simulator's logging database to calculate and store the flows that *would* take place if all reservations were met: this gives us the *forecasted* flows by the Proactive routing algorithm. We also compute and store the *actual* flows measured during the evacuation. By comparing the flows forecasted by the routing algorithm and the actual flows, we can isolate the error introduced by the signs.

As we have mentioned before, the staircases leading to the first floor are the two critical bottlenecks which determine the outcome of the evacuation, so we will focus our analysis on this area of the building. By analysing the collection of routes issued by the Proactive routing algorithm, we can determine the ratio of users assigned to the central staircase r_R. At the end of each simulation, we compute r_T the ratio of users which effectively took the central staircase. The difference between these two ratios is the error introduced by the signs, ε_S. We have computed the value of ε_S for each simulation, and Figure 6.11 shows the empirical probability densities of ε_S, built using the kernel density method[3]

The analysis of Figure 6.11 reveals the following:

1. The centre of every distribution is close to $\varepsilon_S = 0$ with only a small bias for the lower values.

2. Given a number of evacuees, the distribution is narrower for smaller Δt_S values: the error introduced by signs decreases with the cycle time.

3. Given a value of Δt_S, the distribution is narrower as the evacuee headcount increases: the error introduced by signs decreases as the evacuee headcount increases.

Based on observation 1., we conclude that the signs do not create a significant bias towards either staircase. This is most likely because the directions are displayed in a randomised order: as we have explained, maintaining the same order when displaying directions can create bias towards some directions.

We use Figure 6.12 to justify Observations 2 and 3. This figure shows the arrival rate of evacuees in front of the signs on N_s (those which assign them to either staircase). We have also shown a timeline of the time-steps aligned with the main graph: each alternation between yellow and blue corresponds to a new time-step. This makes it simple to visualise the variations in arrival rate over the span of any time-step. The Figure shows that arrival rates globally increase at the beginning of the evacuation, then reach a steady-state and finally decline towards the end as the last evacuees go past. The steady-state regime occurs as soon as evacuees start to form queues in front of the staircases: the edges are saturated and the flow stabilises to the edge's maximal

[3]Despite running 600 simulations, the 20 possible combinations of experimental parameters only leave 30 samples to estimate the distribution of one scenario. Since the original distributions were somewhat rough, we applied additional smoothing in order to emphasise the first and second moments.

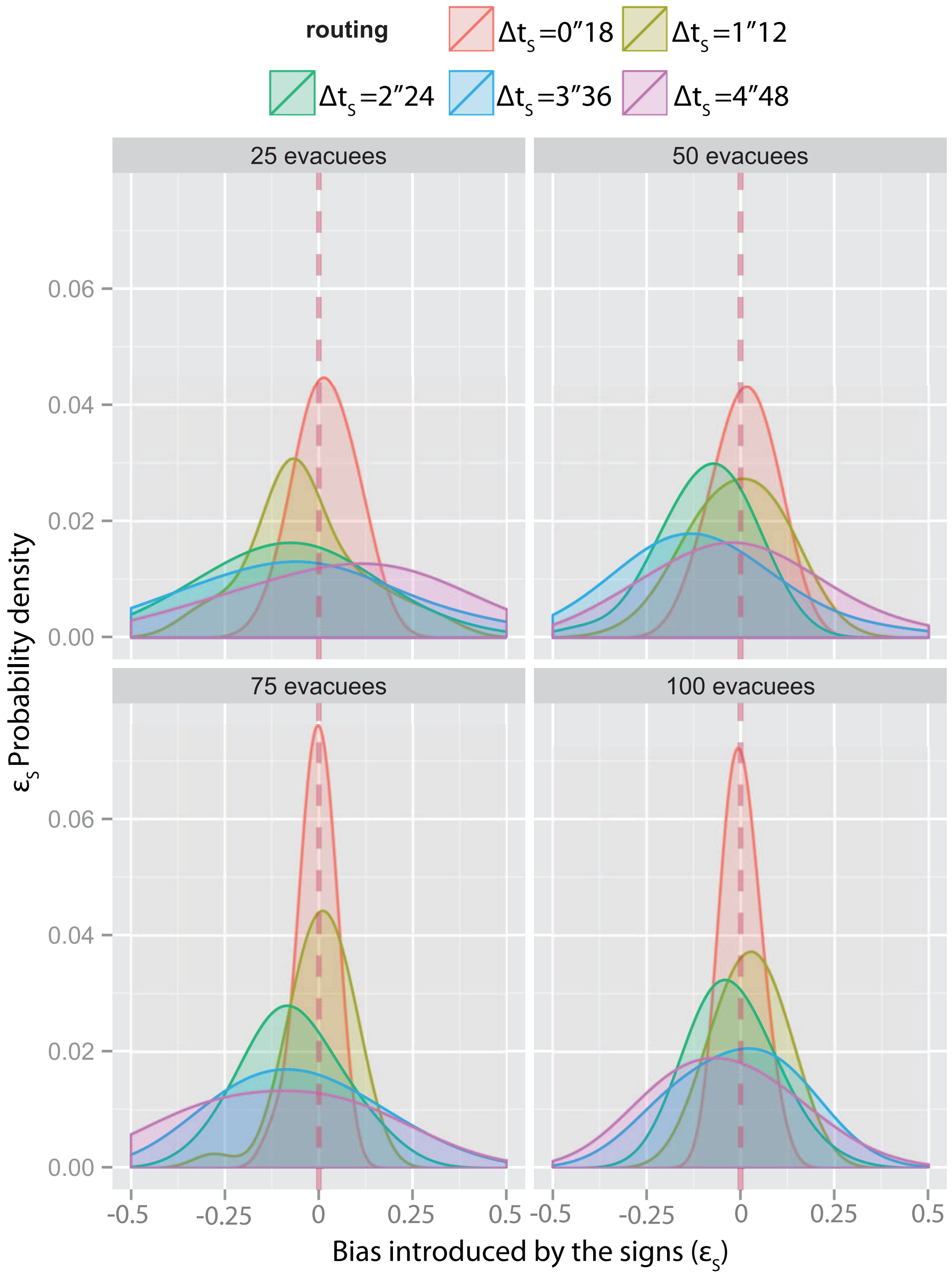


Figure 6.11: Empirical probability density of ε_S, using kernel density estimation method. The graph shows that the error has no significant bias as the distributions are centred on zero. The tall and narrow distributions associated with small Δt_S indicate a low error probability, while the wide distributions associated to higher Δt_S values indicate larger error range and probabilities .

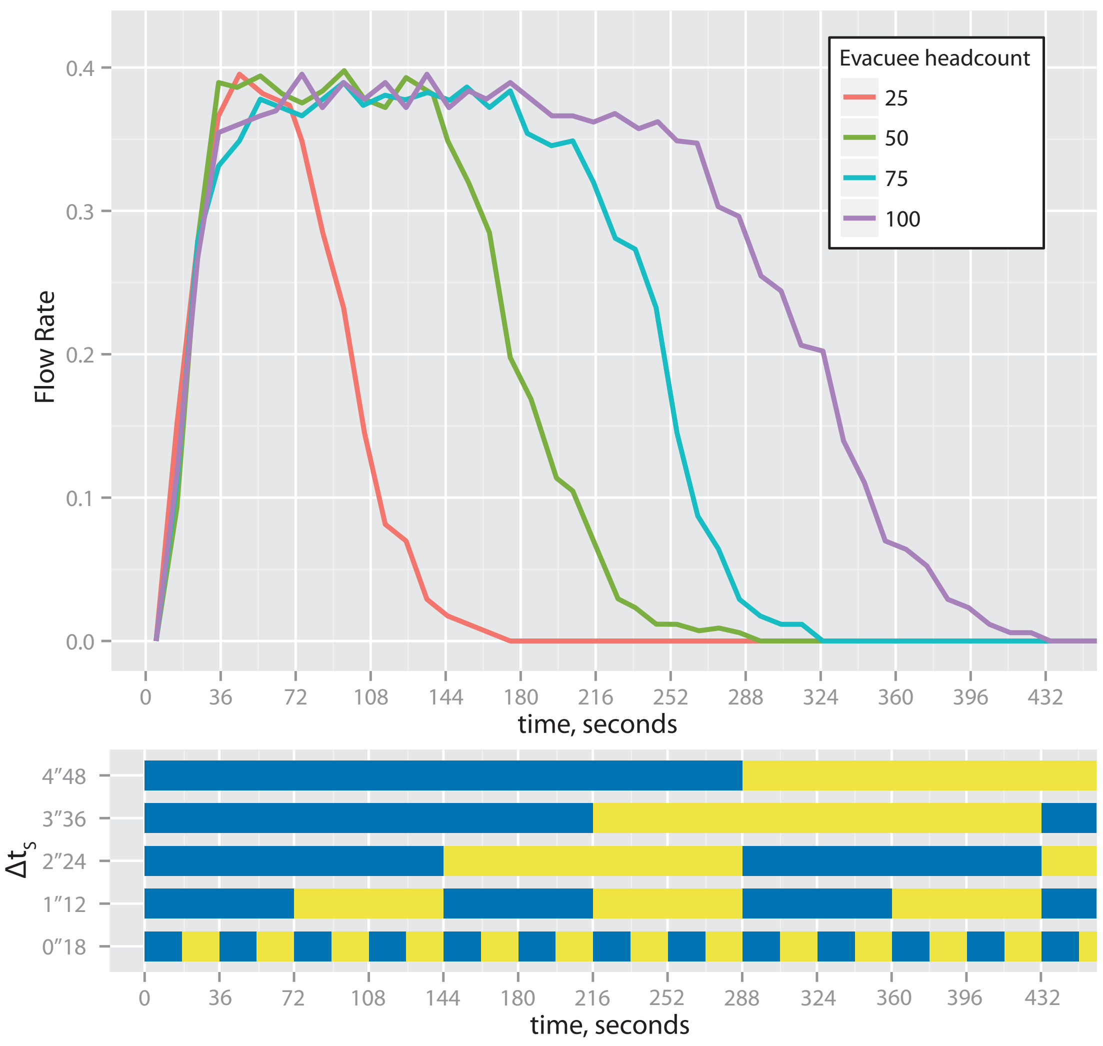


Figure 6.12: Flow of evacuees in front of the signs located on N_s. The timeline below shows the sequence of the dynamic sign's time-steps. The figure shows that for $|F| = 25$, all evacuees will have gone past the signs within a fraction of the first time-step, for any $\Delta t_S \geq 3''36$. In this case, the first direction(s) displayed will be over-assigned, and the directions displayed later on will not be seen by anyone.

processing rate.

We know that the sign's error depends on the integral of the deviation of the arrival rate compared to its mean ($\varepsilon_{n \to e,k}$ in eq. 6.14). In transient states, the integral of the deviation will be smaller if the time-steps are shorter: smaller time-steps break the transient phases into smaller "windows" in which the flow rate's variations remain closer to the mean. Larger time-steps also have higher chances to "straddle" different regimes (i.e. transient and steady-state) where the arrival rates are bound to be different. For

instance, considering $\Delta t_S = 2"24$ and $|F| = 25$ (red curve), the first time-step will encompass the transient "first arrivals" phase, the steady-state phase, and the transient "last departure" phase. Since most arrivals will occur during the short steady-state period, the direction(s) displayed then will be over-assigned, while hardly any evacuee will be left to see the direction(s) displayed towards the end of the time-step, making them under-assigned. In contrast, if we look at smaller time-steps, the variations in arrival rate within a time-step tend to be smaller. We also recall that smaller time-steps give the signs a chance to update the next-hop routing probabilities more often, and therefore provide a more dynamic response. By controlling smaller groups of evacuees, the system better implements the routing algorithm's plan.
Observation 3. is due to the longer duration of the steady-state regime when evacuee headcount increases. Indeed, our system performs best in steady-state when the arrival rate is generally stable, with small random variations.

Evacuation Time

We turn to the building evacuation times (Fig. 6.13) to see the effect the errors introduced by sings have on the system's overall performance. As expected, increases in time-step duration raise median building evacuation times, and widens the samples' distribution. For experiments with 25 – 50 evacuees, the median seems to reach a plateau value from a specific Δt_S value onwards (shown in Table 6.3). These Δt_S values correspond to time-step durations which are longer than the attendance time of the signs on N_s (see Figure 6.12). In these cases, the bulk of the evacuees walk past these signs in the *early* part of the time-step, so effectively most of them only follow the *first* direction(s) displayed, so that the system is effectively static. Unlike the Reactive algorithm whose initial solution is akin to the shortest path, here directions are ordered randomly, which explains why the system's performance is so unpredictable.
As the evacuee headcount increases, we see that the system's performance improves, and the operational limit is reached at increasingly higher Δt_S values. This is because the signs' attendance period also increases: the steady-state regime lasts longer which reduces the sign's error, and the signs can complete more time-steps during their attended period which allows routing probabilities to be updated more often.

	25 Evacuees	50 Evacuees	75 Evacuees	100 Evacuees
Δt_S value	$\geq 2"24$	$\geq 3"36$	$\gg 4"48$	$\gg 4"48$

Table 6.3: Δt_S values from which the system fails to adequately steer the evacuees.

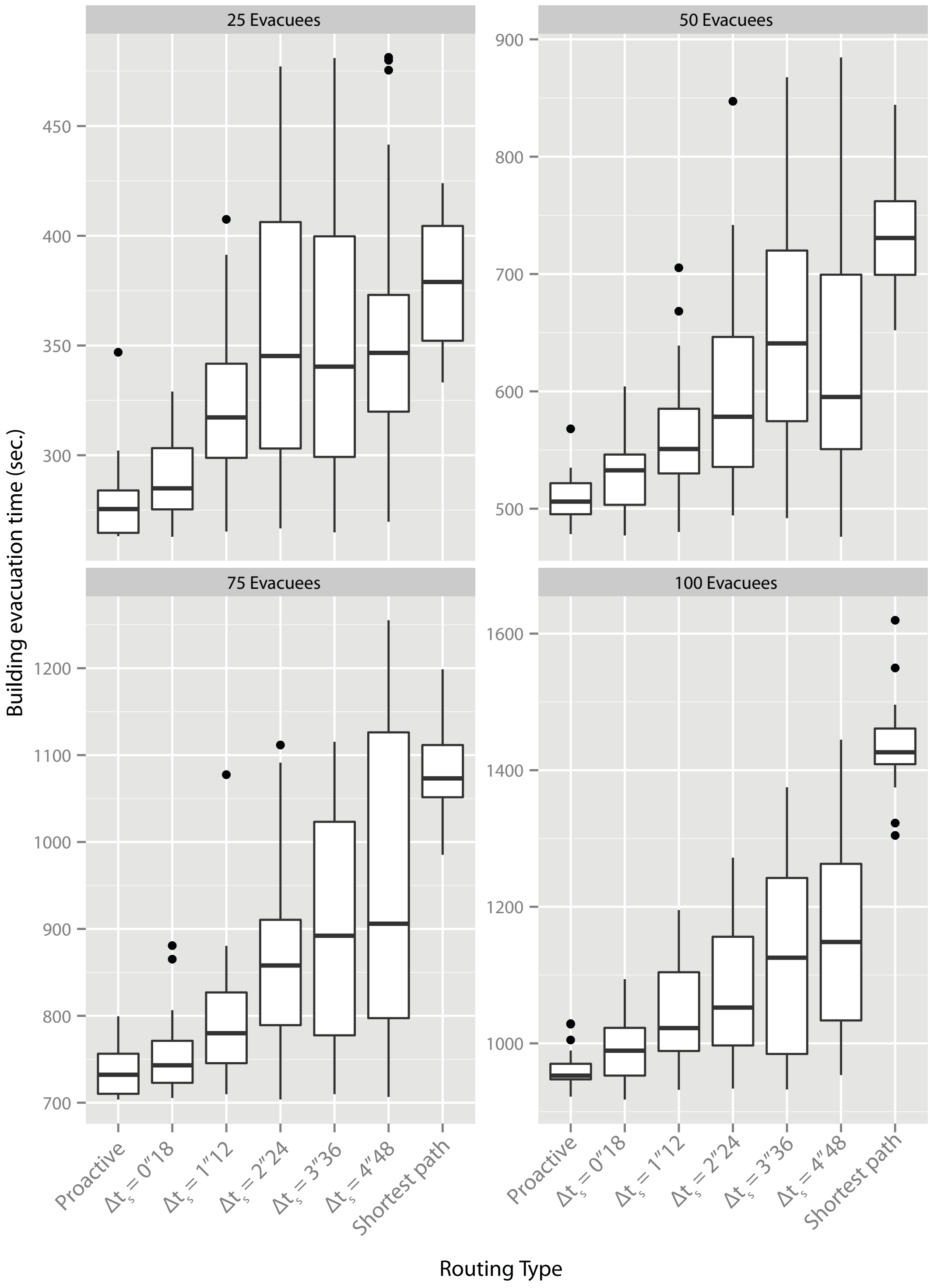


Figure 6.13: Building evacuation times, Proactive routing with dynamic signs. We notice that evacuation times increase as either the time-step duration increases, or the evacuee headcount reduces.

6.3.5 Summary and Discussion

In this section, we have introduced an algorithm to control the signs' display based on the Proactive algorithm's output. The core concept first consists of calculating the ratios of evacuees which continue towards each outbound direction, based on the collection of routes issued by the Proactive algorithm. Then, signs are scheduled to display each direction during a time-step, for a duration proportional to the number of evacuees which should be sent this way. This concept is based on the assumption that routes can be exchanged amongst evacuees as they cross each other, and that the instantaneous rate of arrival is randomly distributed around a mean value. We have reviewed in detail each assumption and predicted the limits they may impose on the system's operation. Finally, we have simulated this system using DBES and analysed the error introduced by the signs, as they apply the Proactive routing algorithm's plan. While the system has proven to be successful, we have seen that the magnitude of error introduced by the signs depends on the time-step duration and the evacuee headcount. Thus our conclusion is similar to that of the Reactive algorithm: the signs must complete a few time-steps during their attended period for the system to perform adequately.

6.4 Summary

In this Chapter, we have focussed on the means to inform the evacuees of the routes issued by the algorithms presented in the previous Chapter. We decided to use dynamic signs, as we believe they are the safest and most intuitive way to guide evacuees in congested environments. Driving dynamic signs from the Reactive algorithm's output is a straightforward process: each node independently controls its sign based on the first-hop of the best path it currently holds in its table. By positioning the dynamic signs slightly before the queuing area, we were able to reduce the effect of the algorithms' oscillations on evacuees.
On the other hand, the Proactive algorithm – being a source-routing algorithm which assigns routes on an individual basis – is fundamentally incompatible with the concept of dynamic signs, since they cannot discriminate between users. To solve this problem, we have assumed that the paths issued by the Proactive algorithm can be re-assigned under certain conditions, and that the time between evacuee arrivals is randomly distributed. Based on these assumptions, we display a direction for a duration proportional to the number of evacuees we wish to send in a direction.
We have designed both sign-driving algorithms so that they do not add any sensory requirement beyond what is needed for the routing algorithm. Both algorithms performed well, but we consistently found that the frequency at which the signs' direction

changes influenced the performance. Theoretically, allowing the signs to change more often improves the algorithm's performance, by providing a finer control over the evacuee population. On the other hand, if the signs switch directions too frequently, evacuees are more likely to distrust the system. Thus defining appropriate Δt_S and Δt_R is effectively an optimisation problem. The lack of available research on the evacuee's compliance to signs, relative to their switching frequency makes it difficult to provide a definitive conclusion on this aspect of the problem.
We have also seen that the systems perform better with larger evacuee populations, for a fixed Δt_S or Δt_R, as the sign's attended periods become longer. The longer the attendance period, the more updates or time-steps a sign can carry out, which again provides a finer control over the evacuee population.

7 Conclusion

7.1 Summary of Problem and Work

The sudden surge in traffic which occurs during emergency building evacuations often leads to widespread congestion on egress paths. Finding a way to route evacuees that minimises the effect congestion has on building evacuation time is a complex and dynamic problem. Currently, building designers elaborate a *static* plan to optimise the evacuation process and increase survival rate in case of hazard outbreak. This static plan is usually designed to produce the best results under worst-case conditions, i.e. when the building is filled to capacity. Yet there are no guarantees this plan will remain optimal in other circumstances, especially under partial occupancy.
In order to overcome the inherent limitations of a static plan and minimise the building evacuation time regardless of the number of evacuees and their location, we have proposed a dynamic system which tailors a flow-optimal evacuation plan to the current scenario.

Evacuee routing We proved that routing evacuees along the shortest path does not always produce optimal results, especially in cases where evacuees are concentrated near a particular egress path. This can be improved if the routing algorithm performs load-balancing on the evacuation paths. This led us to define two methods to estimate the path delay metric:

- The “Reactive” method is based on real-time congestion measurements. While effective at reducing building evacuation times, we found that this method causes route oscillations which will confuse evacuees. Oscillations are due to the delayed feedback between the moment the route is assigned and the moment the resulting increase in congestion occurs. As this is not accounted for, the routing algorithm’s corrections consistently “overshoot” the optimum balance. We have demonstrated that these oscillations can be controlled, but this requires parameters to be tuned to the specific problem, which may be difficult to achieve in real-life applications.
- The “Proactive” approach forecasts congestion by reserving “future capacity” after

each path assignment. Once the routing algorithm foresees a specific egress path becoming saturated, it can preventively start offloading evacuees on an alternative path. This concept has originally been used with Dijkstra's shortest path algorithm, which produces near-optimal results at the cost of large graph searches. We improved on this approach by introducing the CPN routing algorithm and neural networks to reduce the routing overhead and distribute the routing process.

In Chapter 4, we have studied CPN's initial knowledge gathering, and route updating process. In particular, we have shown the importance of the drift parameter (random vs. RNN-guided exploration) on the latency and solution quality. We have also considered how CPN may be distributed to improve the system's resilience and scale to complex buildings or larger occupancies.
Our simulations demonstrate that CPN with path delay metric can reduce evacuation times (compared to a static shortest-path approach), and that both path metric estimation methods presented are effective. We have done so while abstracting the process in which evacuees are informed of their path.

Evacuee guidance In Chapter 6, we have considered the means to inform evacuees of optimal evacuation paths. This aspect of our problem is often dismissed by assuming evacuees carry a personal communication device. Yet we argued that this solution is inadequate, and proposed the use of dynamic signs instead. However, the dynamic signs are incompatible with the Proactive algorithm, since dynamic signs can only indicate the next hop's direction, and cannot discriminate between users. We have therefore developed a novel method to configure dynamic exit signs according to the output of the Proactive routing algorithm. This algorithm aggregates the collection of paths assigned by the routing algorithm; and programs signs to display a direction for a duration which is proportional to the number of evacuees that should be sent this way. We have demonstrated that this system is effective in a typical building office, assuming the evacuees comply with the directions displayed on the signs. We have also implemented a similar evacuee guidance method based on the Reactive routing algorithm and compared the results of both sign-based systems.

7.2 Feasibility

The evacuee assistance system we have presented in this work relies on a framework of components. As we have abstracted most of the hardware components, we will now consider their feasibility and, when possible, issue recommendation for their implementation.

- **Localisation component** While there is currently no method which can accurately measure the density of users in a building, we believe that ongoing developments in RF tracking and video monitoring will provide a suitable solution in the near future. RF-based tracking techniques are possibly the most promising as they can use existing infrastructure, such as wireless network access points. Even if the system can only track a representative subset of the population (i.e. those which have a receiving or tracking device), as long as the total population in the building can be well estimated, the density can be obtained by extrapolating this subset. Furthermore – although this should be verified through experimentation – we believe the system we have presented can tolerate localisation errors in the range of 10 meters, which state-of-the-art systems can meet.

- **Data network** In theory, CPN, the routing algorithm we have proposed, could run on a single central server. However, this would create a single point of failure which could disable the entire evacuee assistance system. Instead, CPN could be distributed over serveral computing devices. This distribution can be done at various scales: from assigning one physical server to each area (or floor) of the building, down to assigning one small computing device to each node in the building graph. While a large-scale distribution may improve scalability and performance in degraded conditions, it will also place higher constraints on the data network, in order to to keep SPs travelling from a node to another at a fast rate. Therefore, we neither recommend the single-server approach, because of associated risks of failure, nor recommend a large-scale distribution, as the cost of the network and node deployment may become be unbearable, and high network performance expectations may also be difficult to meet. As a general recommendation, a middle ground solution where one server is affected to each area of the building could be the best compromise in terms of practicality, performance, robustness and cost-effectiveness. In any case, we believe that the main communication medium should be a *wired* network, to ensure minimal latencies. As the wires of this network may be vulnerable to fire, the network should be designed with a high level of redundancy, or coupled with a wireless network to ensure continued operation in degraded conditions.

- **Computing devices** The requirements on the computing device(s) will also be affected by the level of distribution. Using a single server to run CPN for the entire building means that all SPs will travel "virtually" within the machine: this removes networked communications latencies. However, running CPN on a single computer reduces the ability to send SPs simultaneously from different locations, which impacts the quality of the solution. Furthermore, performing RNN updates is a costly operation which should ideally be distributed over several computers. Thus, the savings in network delays offered by a single-server solution are likely to be lost because of an increased computation time. Although this should be confirmed by conducting further experiments with a realistic network and computer model, distributing the routing process will improve the system's performance and robustness. We believe the routing should be as distributed as possible – it is the latency introduced by the growing data network that will determine to which point the system can be distributed.

- **Dynamic signs** While dynamic signs are virtually non-existent in today's buildings, we believe – since they are only a variation of the standard static exit signs – their adoption will not be a major challenge. Their control could be integrated in the fire monitoring system, or in the building control system (which also typically controls lighting, ventilation, etc.).

At the beginning of this document, we have explained that a static approach is sufficient to evacuate a building filled to capacity in minimal time. Therefore, if the building is expected to be systematically full, our system will be of limited use, as it will generally come up with the same solution as the one displayed by the static approach. Therefore our system is most suited for buildings where occupancy rates and evacuee distributions are highly variable and difficult to predict, such as university buildings, movie complexes, etc.

7.3 Key Conclusions

We will now summarise some key contributions we have made throughout this work.

Reactive or proactive? We have seen that neither a reactive nor proactive approach to the congestion management problem results in an optimal and practical system. The Reactive algorithm does not account for the fact that congestion is related to path assignment. As a result, it requires frequent measurements, regularly issues route corrections, and also tends to oscillate in a way which confuses evacuees. On the other hand, the Proactive algorithm requires little sensory input by comparison, is stable but is also rigid (i.e. cannot adapt to unforseen events) and its performance is reliant on the evacuee motion model's accuracy.

Update frequency of dynamic signs Theoretically, updating the dynamic signs more frequently provides a finer control over the evacuees, which allows the system to reach closer to an optimal solution. On the other hand, allowing the signs to change directions too often reduces evacuee compliance: as evacuees see the sign switching directions many times as they approach it, they might wonder which instruction to follow, or start to distrust the guidance system. Thus the optimal time-step (or update rate) should be a compromise which provides acceptable performance while maintaining evacuee compliance. While we were able to characterise the performance enhancements when the signs switch directions more often, the lack of research on the evacuees' response to dynamic signs prevents us from reaching a definitive conclusion.

Performance in low-occupancy scenarios The systems we have proposed tend to perform better in high evacuee headcount scenarios. It takes longer to evacuate crowded buildings, and this allows the signs to perform more updates or time-steps, and thus better distribute the evacuees over different paths. On one hand, improved performance in crowded buildings is a desirable feature since these scenarios inherently carry a higher risk. On the other hand, our initial objective was to design a system which minimises the evacuation time *regardless* of the distribution and number of evacuees in the building. Yet as the number of evacuees decreases, congestion becomes less and less of a predominant factor. This inherently makes a dynamic congestion management system less and less relevant. In any case, there will always be a lower operational limit (in terms of evacuee headcount) to congestion-optimisation systems.

7.4 Future Work

While presenting the details of our model, we have also listed the parts of the evacuee guidance system we have not modelled (refer to 3.3). Thus developing components which have been "abstracted" constitutes an obvious continuation to this work. In particular, the evacuees' behaviour model should be upgraded to encompass a greater diversity of behaviours – beyond simply complying with the advice given by the system. We have assumed evacuees were cooperative, however in reality, evacuees may act selfishly or selflessly, follow a leader, or simply disregard the system's advice. The system we have proposed is vulnerable to selfish evacuees deciding to take the shortest evacuation route, regardless of the system's advice: while these users will have less distance to walk, their actions will result in cooperative evacuees having to walk through even longer egress routes in order to maintain load balance. As the percentage of such selfish evacuees increases, the shortest route will become increasingly congested, to the point where there are not enough cooperative evacuees to be diverted to other paths (in order to maintain a balance in loads), and the evacuation times will start to increase, and

converge towards the results obtained with a shortest-path routing policy.
Evacuees travelling in groups or following a leader may also pose problem, since they effectively reduce the system's routing resolution. This means the system will not be able to split the group of evacuees to balance loads, instead, the group will choose one of the directions, and create a surge in congestion on that particular path.
The relatively low accuracy of current indoor tracking systems is also likely to affect the real-life system's performance and should be accounted for. Finally, a conclusive validation of the proposed evacuee guidance system should also include simulation scenarios where the infrastructure is damaged.
Beyond addressing the model's limitations listed in Chapter 3, we recommend conducting further work on the topics listed below.

Building topologies We used a typical office building to validate our evacuee guidance systems. On one hand, the relative simplicity of the flow-optimisation solution for this building has allowed us to explore our simulation results in great depth, and precisely determine the root cause of each degradation in performance observed. However, the simplicity of this building may have concealed some limitations or flaws of our system. Therefore our main priority will be to simulate the systems in larger and more complex buildings, such as stadiums or lecture theatres.
We believe the system we have proposed can handle most building topologies; there are, however, some topologies which may limit the system's effectiveness:

- Some topologies may not be suitable for load distribution. For instance, buildings featuring only one main evacuation path would not benefit from our proposed system. The same applies to scenarios where most evacuees are near a particular exit, but alternative exits are too far away to make diverting evacuees worthwhile.
- Sparse areas may also be challenging: if small groups of evacuees start from locations which are far apart, there will not be any "steady-state" phase where arrival rates and routing probabilities are relatively stable. This will require constant adjustments from the system, and accurate measurements or predictions of the evacuee's movements.

On the other hand, we expect our system to perform best in buildings with dense occupancy (such as stadiums, theatres, etc.), and which offer several options in terms of exit routes, and where these routes have comparable lengths.

Dynamic sign synchronisation Evacuees are likely to distrust the guidance system if they see signs displaying conflicting directions, and this will reduce the system's effectiveness. In Reactive routing, nearby nodes may have slightly different views of the network because of CPN's non-optimal performance, and may display contradicting advice. Likewise, the order in which directions are displayed by signs under the Proactive algorithm is randomised, and these signs could also display contradicting directions. Yet an analysis of individual evacuee paths showed that events where evacuees either backtrack or cross each other (walking in different directions) are extremely rare. This may be a feature of our system, or merely owed to the simple building topology. This is another reason why our experiments should be replicated on larger, more complex buildings. If incoherent display occurs, *directional* signs (i.e. displaying different directions based on the incoming direction), may be part of the solution. Otherwise, synchronising the signs using distributed consensus [10], or making them form "platoons" of evacuees [81, 95] may solve the problem.

Hybrid system We have concluded that neither the Reactive nor Proactive routing systems are optimal. In particular, the Proactive system is unable to adapt to unforseen changes in the building map (e.g. areas made impassable because of the spread of a hazard) and changes in the evacuees' mobility model (e.g. evacuees moving slower due to exposure to hazards, lack of visibility, etc.). We believe a hybrid system, combining the Reactive and Proactive algorithms' approach, could lead to a practical and stable, yet *adaptive*, system. However, this approach (effectively based on the Kalman Filter model [85]) may be computationally expensive until a simple method to correct parts of the congestion forecast becomes available.

Bibliography

[1] Fire alarm control panels — Schrack-Seconet:. http://www.schrack-seconet.com/en/products_solutions/fire_alarm/firealarm_control_panels/controlpanel_networking/index.html. Accessed: 2014-01-08.

[2] Guide to safety at sports grounds, fifth edition. department for culture, media and sports.

[3] FIFA Stadium Safety and Security Regulations. Chapter IV: Maximum Safe Capacity of a Stadium, page 39, http://www.fifa.com/mm/document/tournament/competition/51/53/98/safetyregulations_e.pdf. Accessed 3012-09-31.

[4] Fipa: Foundation for intelligent physical agents. http://www.fipa.org/. Accessed: 2013-10-20.

[5] MCG Attendance - 2013 AFL Season. http://www.mcg.org.au/History/Attendances.aspx. Accessed: 2013-09-31.

[6] Jade: Java agent development framework: an open source platform for peer-to-peer agent-based applications. http://jade.tilab.com/. Accessed: 2013-12-20.

[7] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms, and applications.* Prentice hall, 1993.

[8] Jay Aronson. A survey of dynamic network flows. *Annals of Operations Research*, 20(1):1–66, 1989.

[9] Franz Aurenhammer. Voronoi diagrams – evacuation route allocation.a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991.

[10] Arta Babaee and Moez Draief. Distributed binary consensus in dynamic networks. In *Information Sciences and Systems 2013*, volume 264 of *Lecture Notes in Electrical Engineering*, pages 57–65. Springer International Publishing, 2013.

[11] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-event system simulation*. Pearson Education, Upper Saddle River, N.J., fifth edition, 2010.

[12] Matthew Barnes, Hugh Leather, and DK Arvind. Emergency evacuation using wireless sensor networks. In *Local Computer Networks, 32nd IEEE Conference on (LCN 2007)*, pages 851–857. IEEE, 2007.

[13] Richard Bellman. The theory of dynamic programming. Technical report, DTIC Document, 1954.

[14] L.G. Chalmet, R.L. Francis, and P.B. Saunders. Network models for building evacuation. *Fire Technology*, 18(1):90–113, 1982.

[15] Gen-Huey Chen and Yung-Chen Hung. On the quickest path problem. *Information Processing Letters*, 46(3):125 – 128, 1993.

[16] Wen-Tsuen Chen, Po-Yu Chen, Cheng-Han Wu, and Chi-Fu Huang. A load-balanced guiding navigation protocol in wireless sensor networks. In *Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–6. IEEE, 2008.

[17] Christopher Cramer, Erol Gelenbe, and Pamir Gelenbe. Image and video compression. *Potentials, IEEE*, 17(1):29–33, 1998.

[18] A.C. Davies, Jia Hong Yin, and S.A. Velastin. Crowd monitoring using image processing. *Electronics Communication Engineering Journal*, 7(1):37–47, 1995.

[19] P. Derler, E.A. Lee, and A.-S. Vincentelli. Modeling cyber physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.

[20] Antoine Desmet and Erol Gelenbe. Identifying critical sub-systems in the simulation of cyber-physical systems. In *Computer Modeling and Simulation (EMS), 2012 Sixth UKSim/AMSS European Symposium on*, pages 395–400. IEEE, 2012.

[21] Antoine Desmet and Erol Gelenbe. Interoperating infrastructures in emergencies. In *Computer and Information Sciences III*, pages 123–130. Springer, 2013.

[22] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 269:271, 1959.

[23] Nikolaos Dimakis, Avgoustinos Filippoupolitis, and Erol Gelenbe. Distributed building evacuation simulator for smart emergency management. *The Computer Journal*, 53(9):1384–1400, 2010.

[24] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[25] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, April 1972.

[26] Peter Elias, Amiel Feinstein, and Claude Shannon. A note on the maximum flow through a network. *Information Theory, IRE Transactions on*, 2(4):117–119, 1956.

[27] M. Femminella and G. Reali. An experimental system for continuous users tracking in emergency scenarios. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6, 2011.

[28] Avgoustinos Filippoupolitis. *Emergency Simulation and Decision Support Algorithms*. PhD thesis, Imperial College London, Electrical and Electronic Engineering Department, October 2010. URL `http://sa.ee.ic.ac.uk/publications/`.

[29] Avgoustinos Filippoupolitis and Erol Gelenbe. A distributed decision support system for building evacuation. In *Human System Interactions, 2009. HSI'09. 2nd Conference on*, pages 323–330. IEEE, 2009.

[30] Avgoustinos Filippoupolitis, Gokce Gorbil, and Erol Gelenbe. Spatial computers for emergency support. *The Computer Journal*, 2012.

[31] Avgoustinos Filippoupolitis, Laurence Hey, Georgios Loukas, Erol Gelenbe, and Stelios Timotheou. Emergency response simulation using wireless sensor networks. In *Proceedings of the 1st International Conference on Ambient Media and Systems*, Ambi-Sys '08, pages 21:1–21:7, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[32] L. Fleischer and E. Tardos. Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters*, 23(35):71 – 80, 1998.

[33] Lisa K Fleischer. Faster algorithms for the quickest transshipment problem. *SIAM journal on Optimization*, 12(1):18–35, 2001.

[34] DR Ford and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2010.

[35] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[36] Richard L Francis. A 'uniformity principle' for evacuation route allocation. *Journal of Research of National Bureau of Standards*, 86:509–513, 1981.

[37] John Fruin. Designing for pedestrians. *Public Transportation United States*, 1992.

[38] John J Fruin. Pedestrian planning and design. Technical report, Public Transportation, United States, 1971.

[39] L. Fu, D. Sun, and L.R. Rilett. Heuristic shortest path algorithms for transportation applications: State of the art. *Computers And Operations Research*, 33(11):3324 – 3343, 2006. Special Issue: Operations Research and Data Mining.

[40] David Gale. Transient flows in networks. Technical report, DTIC Document, 1958.

[41] E.R. Galea. A general approach to validating evacuation models with an application to exodus. *Journal of Fire Sciences*, 16(6):414–436, 1998.

[42] Ruomei Gao, Constantinos Dovrolis, and Ellen W Zegura. Avoiding oscillations due to intelligent route control systems. In *INFOCOM*. Citeseer, 2006.

[43] E. Gelenbe. Self-aware networks. In *Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on*, pages 227–234, 2011.

[44] E. Gelenbe, Peixiang Liu, and J. Laine. Genetic algorithms for route discovery. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(6):1247–1254, 2006.

[45] Erol Gelenbe. Random neural networks with negative and positive signals and product form solution. *Neural computation*, 1(4):502–510, 1989.

[46] Erol Gelenbe. Learning in the recurrent random neural network. *Neural Computation*, 5(1):154–164, 1993.

[47] Erol Gelenbe. Sensible decisions based on QoS. *Computational management science*, 1(1):1–14, 2003.

[48] Erol Gelenbe. Cognitive packet network. US Patent #6,804,201, 10 2004.

[49] Erol Gelenbe and Michael Gellman. Can routing oscillations be good? the benefits of route-switching in self-aware networks. In *Modeling, Analysis, and Simulation*

of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International Symposium on, pages 343–352. IEEE, 2007.

[50] Erol Gelenbe and Michael Gellman. Oscillations in a bio-inspired routing algorithm. In *Mobile Adhoc and Sensor Systems, 2007. MASS 2007. IEEE Internatonal Conference on*, pages 1–7. IEEE, 2007.

[51] Erol Gelenbe, Michael Gellman, Ricardo Lent, Peixiang Liu, and Pu Su. Autonomous smart routing for network qos. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 232–239. IEEE, 2004.

[52] Erol Gelenbe, Michael Gellman, and George Loukas. An autonomic approach to denial of service defence. In *World of Wireless Mobile and Multimedia Networks, 2005. WoWMoM 2005. Sixth IEEE International Symposium on a*, pages 537–541. IEEE, 2005.

[53] Erol Gelenbe and Ricardo Lent. Power-aware ad hoc cognitive packet networks. *Ad Hoc Networks*, 2(3):205–216, 2004.

[54] Erol Gelenbe, Ricardo Lent, and Arturo Nunez. Self-aware networks and QoS. *Proceedings of the IEEE*, 92(9):1478–1489, 2004.

[55] Erol Gelenbe, Ricardo Lent, and Zhiguang Xu. Design and performance of cognitive packet networks. *Performance Evaluation*, 46(2):155–176, 2001.

[56] Erol Gelenbe, Ricardo Lent, and Zhiguang Xu. Measurement and performance of a cognitive packet network. *Computer Networks*, 37(6):691 – 701, 2001.

[57] Erol Gelenbe, Ricardo Lent, and Zhiguang Xu. Towards networks with cognitive packets. In *Performance and QoS of next generation networking*, pages 3–17. Springer, 2001.

[58] Erol Gelenbe and Guy Pujolle. *Introduction to queueing networks*, chapter 1. Wiley Chichester, second edition, 1987.

[59] Erol Gelenbe, Georgia Sakellari, and Maurizio D'arienzo. Admission of qos aware users in a smart network. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 3(1):4, 2008.

[60] Erol Gelenbe, Esin Seref, and Zhiguang Xu. Simulation with learning agents. *Proceedings of the IEEE*, 89(2):148–157, 2001.

[61] Erol Gelenbe and Fang-Jing Wu. Large scale simulation for human evacuation and rescue. *Computers & Mathematics with Applications*, 64(12):3869 – 3880, 2012. Theory and Practice of Stochastic Modeling.

[62] Erol Gelenbe and Fang-Jing Wu. Future research on cyber-physical emergency management systems. *Future Internet*, 5(3):336–354, 2013.

[63] Erol Gelenbe, Zhiguang Xu, and Esin Seref. Cognitive packet networks. In *Tools with Artificial Intelligence, 1999. Proceedings. 11th IEEE International Conference on*, pages 47–54. IEEE, 1999.

[64] Michael Gellman. *QoS Routing for Real-time Traffic*. PhD thesis, Imperial College London, Electrical and Electronic Engineering Department, 2007. URL `http://san.ee.ic.ac.uk/publications/`.

[65] Michael Gellman and Peixiang Liu. Random neural networks for the adaptive control of packet networks. In *Artificial Neural Networks–ICANN 2006*, pages 313–320. Springer, 2006.

[66] Betsy George and Shashi Shekhar. Time-aggregated graphs for modeling spatio-temporal networks. In *Journal on Data Semantics XI*, volume 5383 of *Lecture Notes in Computer Science*, pages 191–212. Springer Berlin Heidelberg, 2008.

[67] Gokce Gorbil. *Opportunistic Communications for Emergency Support*. PhD thesis, Imperial College London, Electrical and Electronic Engineering Department, January 2013. URL `http://san.ee.ic.ac.uk/publications/`.

[68] Gokce Gorbil and Erol Gelenbe. Resilient emergency evacuation using opportunistic communications. In *Computer and Information Sciences III*, pages 249–257. Springer London, 2013.

[69] AK Gupta and Pankaj K Yadav. SAFE-R: a new model to study the evacuation profile of a building. *Fire Safety Journal*, 39(7):539 – 556, 2004.

[70] Bruce Hajek and Richard G. Ogier. Optimal dynamic routing in communication networks with continuous traffic. *Networks*, 14(3):457–487, 1984.

[71] H. W. Hamacher and S. Tufekci. On the use of lexicographic min cost flows in evacuation modeling. *Naval Research Logistics (NRL)*, 34(4):487–503, 1987.

[72] Horst W Hamacher and Stevanus A Tjandra. Mathematical modelling of evacuation problems–a state of the art. Technical report, Fraunhofer Institute for Industrial Mathematics ITWM, 2001.

[73] Qing Han. Managing emergencies optimally using a random neural network-based algorithm. *Future Internet*, 5(4):515–534, 2013.

[74] Dirk Helbing, Illes Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[75] Dirk Helbing, Illes J Farkas, Peter Molnar, and Tamás Vicsek. Simulation of pedestrian crowds in normal and evacuation situations. *Pedestrian and evacuation dynamics*, 21:21–58, 2002.

[76] Dirk Helbing, Anders Johansson, and Habib Zein Al-Abideen. Dynamics of crowd disasters: An empirical study. *Physical review E*, 75(4):046109, 2007.

[77] Simo Helivaara, Juha-Matti Kuusinen, Tuomo Rinne, Timo Korhonen, and Harri Ehtamo. Pedestrian behavior and exit selection in evacuation of a corridor an experimental study. *Safety Science*, 50(2):221 – 227, 2012.

[78] Laurence Hey. *Simplified Adaptive Routing and its Impact on Quality of Service and Quality of Information*. PhD thesis, Imperial College London, Electrical and Electronic Engineering Department, 2009. URL `http://san.ee.ic.ac.uk/publications/`.

[79] Laurence A Hey. Power aware smart routing in wireless sensor networks. In *Next Generation Internet Networks, 2008. NGI 2008*, pages 195–202. IEEE, 2008.

[80] Laurence A. Hey. Reduced complexity algorithms for cognitive packet network routers. *Computer Communications*, 31(16):3822 – 3830, 2008. Performance Evaluation of Communication Networks (SPECTS 2007).

[81] John A Hillier and Richard Rothery. The synchronization of traffic signals for minimum delay. *Transportation Science*, 1(2):81–94, 1967.

[82] Bruce Hoppe and Éva Tardos. Polynomial time algorithms for some evacuation problems. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, pages 433–441, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[83] Bruce Hoppe and Eva Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25(1):pp. 36–62, 2000.

[84] John J. Jarvis and H. Donald Ratliff. Note – some equivalent objectives for dynamic network flow problems. *Management Science*, 28(1):106–109, 1982.

[85] Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[86] Jeff L. Kennington and Richard V. Helgason. *Algorithms for Network Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1980.

[87] Sangho Kim, Betsy George, and Shashi Shekhar. Evacuation route planning: scalable heuristics. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, page 20. ACM, 2007.

[88] T. M. Kisko, R. L. Francis, and C. R. Nobel. EVACNET4 User's guide. Technical report, University of Florida, 1998.

[89] Thomas M Kisko and Richard L Francis. EVACNET+: a computer program to determine optimal building evacuation plans. *Fire Safety Journal*, 9(2):211–220, 1985.

[90] ED Kuligowski. Review of 28 egress models. In *Kuligowski (Eds.), Proceedings of the Workshop on Building Occupant Movement during Fire Emergencies*, pages 68–90, 2004.

[91] Erica D Kuligowski and Richard D Peacock. A review of building evacuation models. Technical report, NIST, 2005.

[92] SM Lo, Hui-Chi Huang, Peng Wang, and KK Yuen. A game theory based exit selection model for evacuation. *Fire Safety Journal*, 41(5):364–369, 2006.

[93] Georgios Loukas and Stelios Timotheou. Connecting trapped civilians to a wireless ad hoc network of emergency response robots. In *Communication Systems, 2008. ICCS 2008. 11th IEEE Singapore International Conference on*, pages 599–603. IEEE, 2008.

[94] Qingsong Lu, Betsy George, and Shashi Shekhar. Capacity constrained routing algorithms for evacuation planning: A summary of results. In *Advances in Spatial and Temporal Databases*, volume 3633 of *Lecture Notes in Computer Science*, pages 291–307. Springer Berlin Heidelberg, 2005.

[95] John T Morgan and John DC Little. Synchronizing traffic signals for maximal bandwidth. *Operations Research*, 12(6):896–912, 1964.

[96] Roya Olyazadeh. Evaluating dynamic signage for emergency evacuation using an immersive video environment. Master's thesis, Instituto Superior de Estatstica e Gesto de Informao (ISEGI), 2013.

[97] Xiaoshan Pan, Charles S Han, Ken Dauber, and Kincho H Law. A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations. *AI & Society*, 22(2):113–132, 2007.

[98] T. Pongthawornkamol, S. Ahmed, K. Nahrstedt, and A. Uchiyama. Zero-knowledge real-time indoor tracking via outdoor wireless directional antennas. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 253–261, 2010.

[99] VM Predtechenskii and Anatoliĭ Ivanovich Milinskiĭ. *Planning for foot traffic flow in buildings.* National Bureau of Standards, US Department of Commerce, and the National Science Foundation, Washington, DC, 1978.

[100] Georgia Sakellari. *Maintaining Quality of Service and Reliability in Self-Aware Networks.* PhD thesis, Imperial College London, Electrical and Electronic Engineering Department, January 2009. URL `http://san.ee.ic.ac.uk/publications/`.

[101] Georgia Sakellari. The cognitive packet network: a survey. *The Computer Journal*, 53(3):268–279, 2010.

[102] Georgia Sakellari and Erol Gelenbe. Adaptive resilience of the cognitive packet network in the presence of network worms. In *Proceedings of the NATO Symposium on C3I for Crisis, Emergency and Consequence Management*, pages 11–12, 2009.

[103] Georgia Sakellari and Erol Gelenbe. Demonstrating cognitive packet network resilience to worm attacks. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 636–638, New York, NY, USA, 2010. ACM.

[104] Andreas Schadschneider, Wolfram Klingsch, Hubert Klüpfel, Tobias Kretz, Christian Rogsch, and Armin Seyfried. Evacuation dynamics: Empirical results, modeling and applications. In *Extreme Environmental Events*, pages 517–550. Springer, 2011.

[105] Alexander Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, 2002.

[106] T.J Shields and K.E Boyce. A study of evacuation from large retail stores. *Fire Safety Journal*, 35(1):25 – 49, 2000.

[107] John H Sorensen. When shall we leave? factors affecting the timing of evacuation departures. *International Journal of Mass Emergencies and Disasters*, 9(2):153–165, 1991.

[108] Peter A Thompson and Eric W Marchant. A computer model for the evacuation of large building populations. *Fire Safety Journal*, 24(2):131–148, 1995.

[109] Yu-Chee Tseng, Meng-Shiuan Pan, and Yuen-Yung Tsai. Wireless sensor networks for emergency navigation. *Computer*, 39(7):55–62, 2006.

[110] Armin Veichtlbauer and Thomas Pfeiffenberger. Dynamic evacuation guidance as safety critical application in building automation. In *Critical Information Infrastructure Security*, volume 6983 of *Lecture Notes in Computer Science*, pages 58–69. Springer Berlin Heidelberg, 2013.

[111] D.T. Vemula, Xunyi Yu, and A. Ganz. Real time localization of victims at an emergency site: Architecture, algorithms and experimentation. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 1703–1706, 2009.

[112] Hui Xie, Lazaros Filippidis, Edwin R Galea, Darren Blackshields, and Peter J Lawrence. Experimental analysis of the effectiveness of emergency signage and its implementation in evacuation simulation. *Fire and Materials*, 36(5-6):367–382, 2012.

[113] Hui Xie, Edwin Galea, and Peter Lawrence. Experimental study of the effectiveness of dynamic signage system. In *Interflam 2013: 13th International Fire Science and Engineering Conference*. Interscience Communications, 2013.

8 Appendix

8.1 DBES Project management Experience

This research project gave us the opportunity to manage a somewhat large experimental process involving over 1,000 simulations. This gave us some valuable experience in managing large sets of data, and also managing a software development project. In this section, we reflect on some of the challenges we faced, and how we solved them.

As our research project advanced, the experimental data accumulated to the point of being difficult to retrieve, process and backup. As we constantly adapted and developed new modules, large amounts of time were invested to become familiar with the intricate details of the DBES code which is largely purpose-built "custom code". In the course of this research, we constantly strived to streamline the data management and the code development process.

8.1.1 Distribution

While DBES is able to breakdown, distribute and coordinate a very large simulation over a pool of computers, we did not use this feature. Indeed, the relative small-scale of our simulations meant that they could be run by a single machine. Whenever possible, running simulations over a single machine is the preferred option since it reduces the amount of network communications that take place, which reduces simulation run-time. Yet in order to run over a thousand simulations required for our project in a reasonable amount of time, we used several computers to run, independently, a subset of the batch of simulations.
In order to distribute and coordinate the joint effort, each machine in the pool of computers is assigned a "job sheet" which describes the simulations it is expected to run. As our experimental process included many re-runs with modifications in the code, the repeated process of building job sheets gradually became tedious and inefficient: it was impossible to add or remove jobs "on-the-fly" or trigger a re-run. The efficiency of this method is also limited: the faster computers would often complete their task early and wait, idle, for the slower ones to complete their assignment.

In order to to improve this process, we centralised the job assignment process using a single “job sheet”. Our philosophy was to assign jobs to computers one at a time, on a first-come, first-served basis: as soon as a machine completes a simulation, it looks up the central job sheet and assigns itself to the next unallocated simulation job. This process potentially involves concurrent read and write operations to the job file, from different processes, and is not supported by operating systems. Instead of developing custom access-control code, we used a MySQL database and leveraged its native support of concurrent operations. Using a database instead of a log file also presented many additional advantages: ability to add or remove jobs during the process, simple retrieval of the next unallocated job over the network using database connection libraries, and a simple SQL query to fetch the next unallocated job.

8.1.2 Data Management and Processing

Our experimental protocol requires running each simulation scenario at least 20 times with some randomised parameters. The different scenarios for a given experiment comprise of 4 evacuee headcount values, and usually a variety of experimental parameters related to the assistance system. As a result, a single experiment generally involves a large number of simulations: generally several hundred. It also became apparent that building evacuation times only told “half of the story”: in order to gain a deep insight into the system’s performance, we would have to analyse the path taken by each evacuee, its length, and the evacuee’s average speed at any point in time. Storing this information, over the course of this project, accounts for over 2 million records containing an {evacuee – location – time} relationship, which accounts for approximately 360mb of data.

Traditionally, the outcome of DBES simulations were written to log files, which were then parsed and processed using a script. While this method is convenient for small data sets, it does not scale well to larger projects and complex analysis. A limitation of this approach is that log files can only be written to by one process at a time, yet DBES is an agent-based simulator where each evacuee is run as a separate process. This means each evacuee agent must create its own log file, and at the end of a batch of simulations, numerous log files are scattered over all machines in the pool, and require a complex merging process. We replaced this system with a central database dedicated to the collection of experimental results. This altogether removed the need to merge the information at the end of simulation runs.

An added benefit of the database’s support of concurrent read/write operations is that we could generate preliminary results without having to wait for the process to complete, allowing us to decide whether to pursue the experiment, or to reduce or increase the number of runs to reach an adequate level of confidence in the results. As our experiments spanned over several months, the format used in log files constantly changed as we added and removed output variables. Handling this multitude of different log file formats required a complex, time-consuming adaptive parsing script. The use of databases allowed us to constrain the format of the information, and guarantee a query would return a valid result regardless of how long ago the experiments had been run. Finally, keeping record of the information became problematic as the number of log files accumulated. In contrast, a single database holds all our data, and assigns a unique ID number to each simulation job, which is replicated on each related information record. This makes relating any piece of information to the original simulation job trivial.

8.1.3 Use of "Off-The-Shelf" Solutions

DBES is mainly built upon"custom-made" or purpose-built software components. Thus each modification or addition to the simulator involves spending a significant amount of time familiarising oneself with custom-built routines. In many cases, "off-the-shelf" solutions exist to address these problems, which allow the developer to focus on the task at hand, and spend less time developing low-level routines. We will present three case studies where we replaced "custom" components with off-the-shelf components and achieved sizeable time savings.

Centralising and multiple indexing We observed that most of the agent communications were "many-to-one": evacuee agents mostly communicate with their "area agent": an agent which manages the simulation clock and centralises the information relative to entities on its area of responsibility. Essentially, the area agent is a data aggregator which stores and retrieves evacuee-related information and despatches simulation events to the relative agents. In many cases, the area agent receives queries which require processing information based on different indices. For instance, a request may query the location of a particular agent, or conversely, query a list of agents at a given location. Thus to achieve optimal search performance, the {`Evacuee ID - Location ID`} relationship table should ideally be sorted both by `evacuee ID` and by `Location ID`. This can be achieved by maintaining two copies of the array, each sorted according to one of the metrics. Instead, we opted for a MySQL database which handles multiple indexing transparently. This improved the simulation time, while reducing the amount of code in the simulator. As agents are able to directly issue SQL queries to the database, the agent messaging traffic is reduced, while the fact that the SQL query specifies the format of the reply guarantees the reply will be understood.

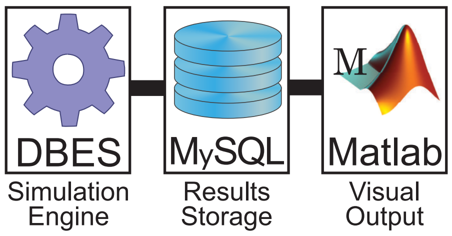
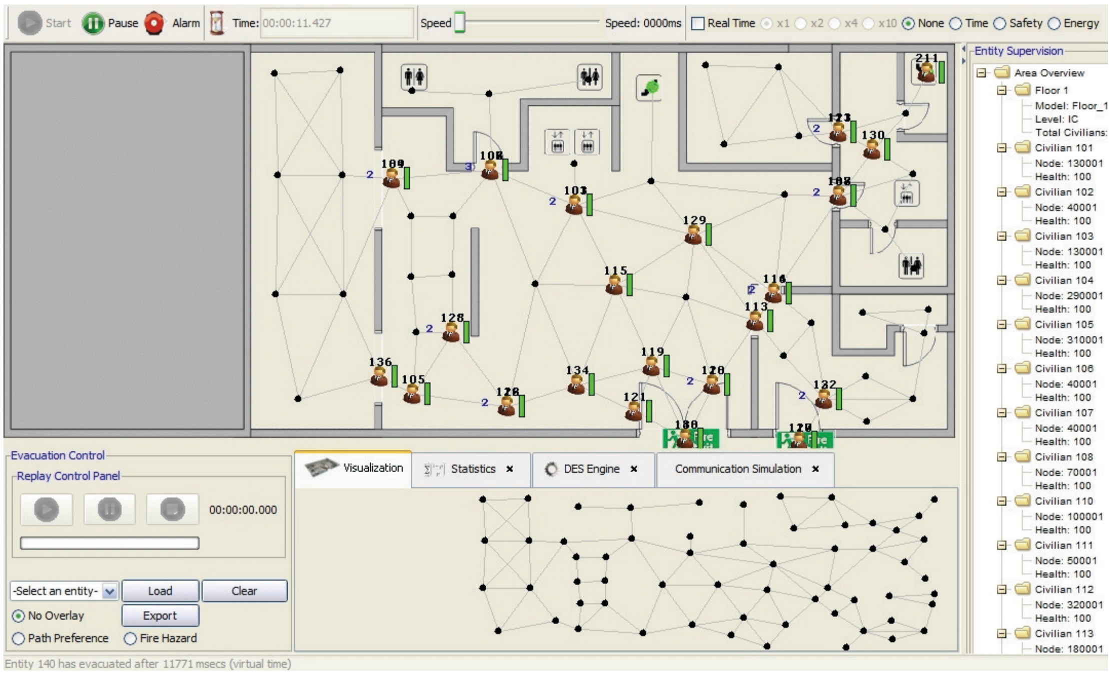


Figure 8.1: Diagram showing the components which have been "outsourced" from DBES: a MySQL database provides data storage and MATLAB is used for real-time visualisation.

Post-processing The log files which DBES outputs have to be parsed and processed by a script. Using a database removes the need to perform any parsing, and its querying and processing capabilities mean that most of the data processing can be carried out by the database itself. Indeed, beyond counting or calculating sums, the MySQL database we use can also calculate arithmetic means and standard deviations. We were also able to execute complex queries with minimal effort while avoiding complex nested `for` loops.

User interface DBES' graphical user interface is a purpose-built component which displays the location of every evacuee as the simulation progresses (see the screen capture on Figure 8.2a). This visual tool proved essential to understand where the congestion takes place during the evacuation, in order to optimise or fine-tune the routing algorithm. However this can result in cluttered visualisations, especially with high evacuee headcounts. In such cases, visualising the *flow* levels on each edge gives a better understanding of the process. DBES' interface is not built for this purpose, yet our project demanded a flow-oriented visualisation tool. Instead of developing another custom visualisation module, we used off-the-shelf components to provide a working solution in very little time. Our approach uses MATLAB's 3-D plotting abilities to provide a representation of the graph, where the edge's line thickness and colour represents queue levels and instantaneous flows, see Fig 8.2b for an example. This visualisation method now provides us with a clearer picture of the congestion in the building. We also used MATLAB's database toolbox to interface with the results database, and refresh the display at regular intervals to visualise the evacuation being simulated in real time.



Time: 00:00:11.427 Speed: 0000ms ☐ Real Time ◯ x1 ◯ x2 ◯ x4 ◯ x10 [✓ None] ◯ Time ◯ Safety ◯ Energy

00:00:00.000

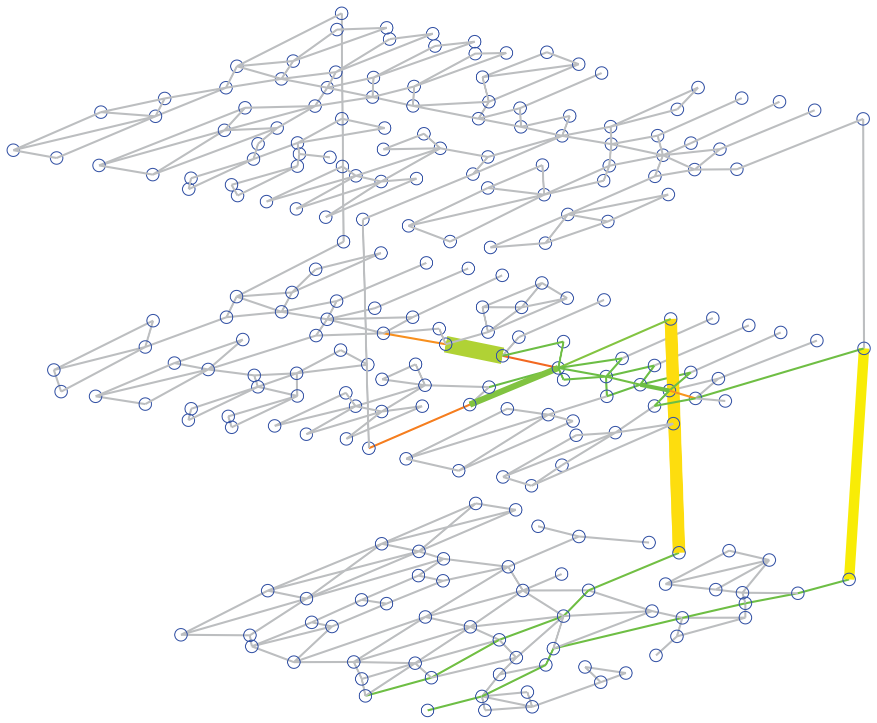
-Select an entity-

[✓ No Overlay] ◯ Path Preference ◯ Fire Hazard

Area Overview
- Floor 1
  - Model: Floor_1
  - Level: IC
  - Total Civilians:
- Civilian 101
  - Node: 130001
  - Health: 100
- Civilian 102
  - Node: 40001
  - Health: 100
- Civilian 103
  - Node: 130001
  - Health: 100
- Civilian 104
  - Node: 290001
  - Health: 100
- Civilian 105
  - Node: 310001
  - Health: 100
- Civilian 106
  - Node: 40001
  - Health: 100
- Civilian 107
  - Node: 40001
  - Health: 100
- Civilian 108
  - Node: 70001
  - Health: 100
- Civilian 110
  - Node: 100001
  - Health: 100
- Civilian 111
  - Node: 50001
  - Health: 100
- Civilian 112
  - Node: 320001
  - Health: 100
- Civilian 113
  - Node: 180001

Entity 140 has evacuated after 11771 msecs (virtual time)

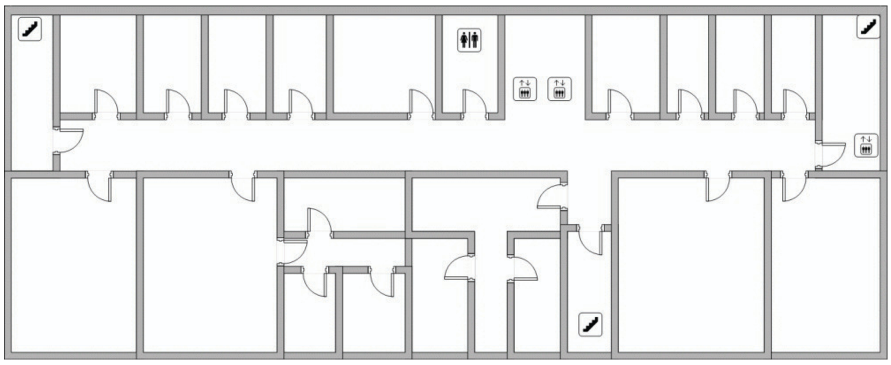
(a) DBES' built-in interface.

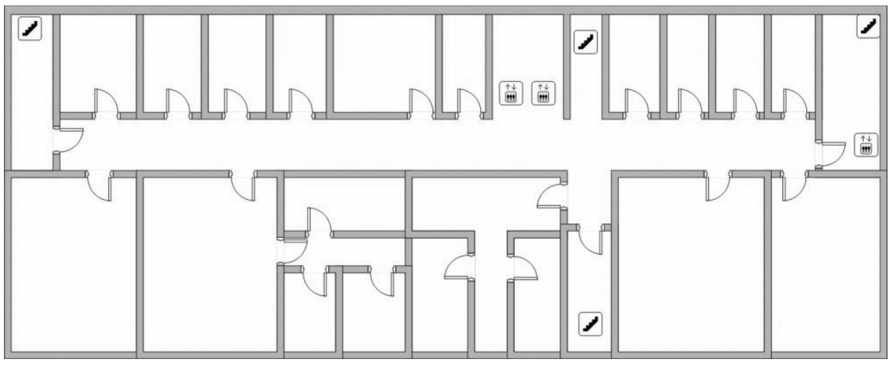


(b) Flow-oriented visualisation interface: line thickness corresponds to queue levels, colours indicate increasing or decreasing flow levels.

Figure 8.2: Comparison of DBES interfaces

8.2 Additional Figures and Illustrations


(a) 3rd floor


(b) 2nd floor

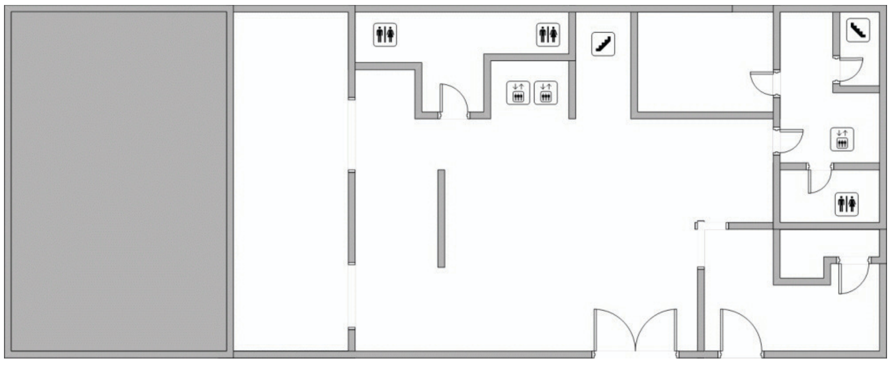

(c) 1st floor

Figure 8.3: Building functional layout

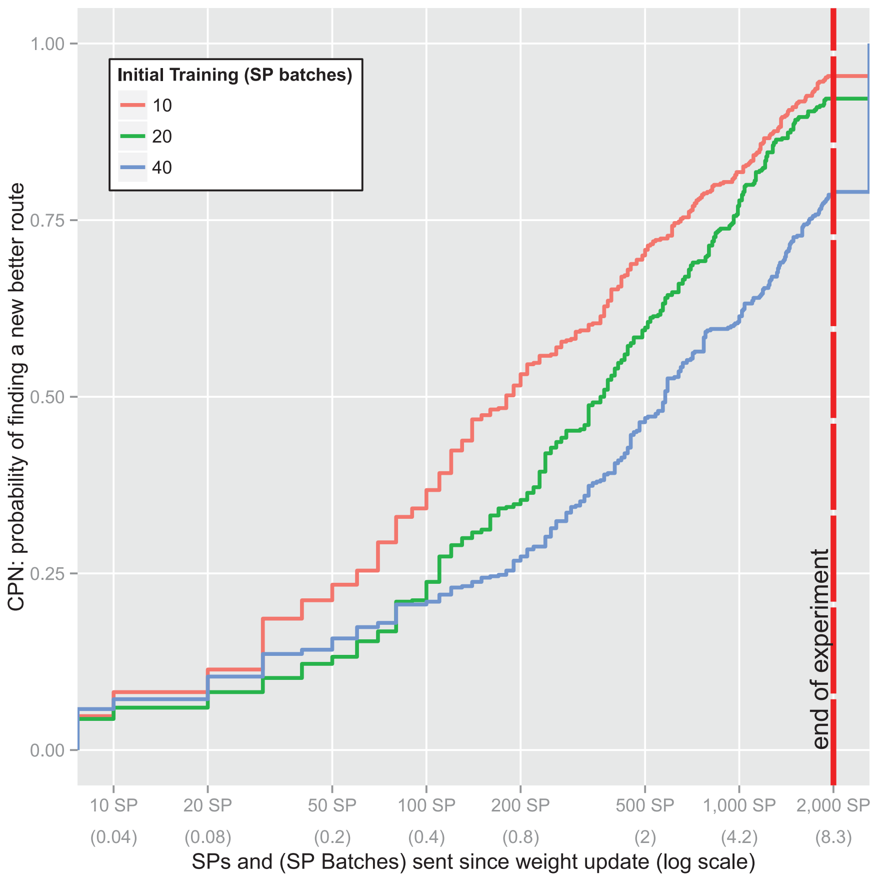


Figure 8.4: Probability of CPN switching to a better path, based on the number of Smart Packets sent after the modification in original route cost. Each curve corresponds to a different amount of initial training, in SP batches.

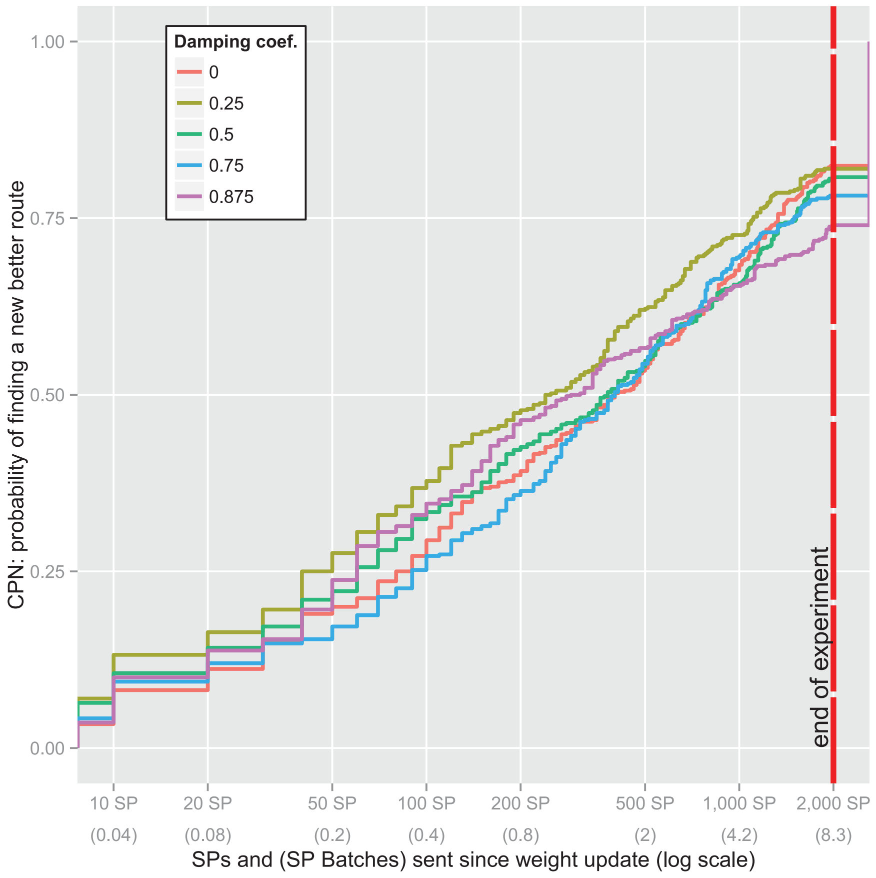


Figure 8.5: Probability of CPN switching to a better path, based on the number of Smart Packets sent after the modification in original route cost. Each curve corresponds to a damping coefficient value.