



THE UNIVERSITY *of* EDINBURGH

This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

- This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated.
- A copy can be downloaded for personal non-commercial research or study, without prior permission or charge.
- This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author.
- The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.
- When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

Topology Based Global Crowd Control

Adam Barnett



Doctor of Philosophy

Institute of Perception, Action and Behaviour

School of Informatics

University of Edinburgh

2014

Abstract

We propose a method to determine the flow of large crowds of agents in a scene such that it is filled to its capacity with a coordinated, dynamically moving crowd. Our approach provides a focus on cooperative control across the entire crowd. This is done with a view to providing a method which animators can use to easily populate and fill a scene. We solve this global planning problem by first finding the topology of the scene using a Reeb graph, which is computed from a Harmonic field of the environment. The Maximum flow can then be calculated across this graph detailing how the agents should move through the space. This information is converted back from the topological level to the geometric using a route planner and the Harmonic field. We provide evidence of the system's effectiveness in creating dynamic motion through comparison to a recent method. We also demonstrate how this system allows the crowd to be controlled globally with a couple of simple intuitive controls and how it can be useful for the purpose of designing buildings and providing control in team sports.

Acknowledgements

I would like to acknowledge my family and friends for all their help and support. Also my supervisor Taku Komura and all of my co-students: Benjamin Rosman, Joe Henry, Peter Sandilands, Vladimir Ivan, Xi Zhao, Hubert Shum, He Wang and Myung Geol Choi for all of their advice, guidance and assistance throughout.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Adam Barnett)

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	Problem Definition and Goals	2
1.2.1	Problem Definition	2
1.2.2	Goals	4
1.3	Key Issues	4
1.4	Publications	5
1.5	Methodology Overview	5
1.6	Thesis Structure	6
2	Related Work	7
2.1	Local Systems	8
2.1.1	Geometric Systems	8
2.1.2	Particle Based Systems	10
2.1.3	Agent Based Systems	11
2.1.4	Crowd Control from Data	13
2.2	Global Systems	16
2.2.1	Field Based Systems	17
2.2.2	Topology Based Systems	19
3	Harmonic Fields and Topology	23
3.1	Harmonic Field	24
3.1.1	Use of the Harmonic field	24
3.1.2	Computing the Harmonic field	25
3.1.3	Finding the Harmonic field Gradient	27
3.1.4	Flowlines	28
3.2	Reeb Graph	31

3.2.1	Related Reeb Graph Work	32
3.2.2	Computing the Reeb Graph	32
4	Maximum Flow	37
4.1	Finding the Capacities	38
4.1.1	Computing Iso-flowlines	38
4.2	Computing the Maximum Flow	43
4.3	Problems with Pure Maximum Flow	44
5	Directing the Agents	47
5.1	Route Planning Problems	48
5.2	Route Planning Methodology	49
5.2.1	Using the RouteTrees	49
5.2.2	Stepping Through the Graph	52
5.2.3	Ordering the Edges	54
5.2.4	Route Planning Pseudo-code	55
5.2.5	Crossing Streams of Agents	56
5.3	Providing Paths	59
5.3.1	Connecting partial paths	60
5.4	Applying the Paths	61
5.5	Congestion and Cooperation	64
6	Experimental Results	67
6.1	Real World Examples	67
6.2	Computational Costs	71
6.3	Comparison	74
6.4	Cooperation and Congestion	78
6.5	Realism of the system	81
6.6	Comparing Reeb Graph to Medial Axis	84
6.6.1	Medial Axis Topology Comparison	85
6.6.2	Medial Axis Capacity Comparison	88
6.6.3	Medial Axis Comparison Conclusion	91
7	Additional Applications	93
7.1	Evacuation Design	93
7.2	Related Work	94

7.2.1	Pathing Analysis	96
7.2.2	Signage Analysis	97
7.2.3	Evacuation System Experiments	99
7.2.4	Evacuation System Conclusion	102
7.3	Football Controller	102
7.3.1	Evaluating the Game State	104
7.3.2	Controlling the Attackers	106
7.3.3	Controlling the Defenders	109
7.3.4	Football System Experiments	110
7.3.5	Football System Conclusion	113
8	Conclusion	117
8.1	Contributions	117
8.2	The Bigger Picture	117
8.3	Future Work	118
	Bibliography	121

Chapter 1

Introduction

1.1 Motivation

Crowds and the associated behaviour of large groups of people emerge from the localised interaction of individual humans. They occur when a large number of people have goals which cause them to intersect, using local collision avoidance and particular crowd specific behaviours, all at once. Generally, simulation of these crowd situations for graphics focuses on a similar local agent based perspective, with the primary metrics of success or failure being the collisions between agents, such as in Kapadia et al. (2011a), or the realism of the scene produced, measured by comparing emergent artefacts to those found in actual crowds as in Helbing and Molnár (1995). Methods which perform well at these tests are poor at producing scenes where a crowd moves in a dynamic fashion. Here we define a dynamic crowd as consisting of a constant and high degree of motion across all of the agents without congestion or queuing. Examples of these types of situations are found in many animations and include such scenes as panicked people rushing away from a disaster or an army charging together towards a specific target. Such scenes focus on the motion of the crowd as a whole rather than local realism and they require additional cooperation between the agents either because the crowd are cooperating, as in an army, or to avoid congestion (which would detract from the dynamic motion of the crowd and be undesirable for the overall composition of the scene), as in a fleeing group of panicked people. We are interested in taking a different approach which utilises global control methods to provide a system which prioritises control and ease of use over realism for the purposes of creating such scenes. This is a worthwhile direction for a few reasons:

1. A method which can fill a scene with a dynamic crowd with minimal control

inputs would have great utility in crowd animation for producing scenes where the intended focus is on the motion of the crowd.

2. There are also currently few methods (discussions of which can be found in Section 2.2.2) which tackle the problem of producing the motion of coordinated crowds or groups of agents, particularly over the entire course of the motion.
3. We believe that providing additional controls for animators to produce crowds and alter their behaviour at a crowd-wide level should be a higher priority and would represent a useful step in generating crowd scenes.

1.2 Problem Definition and Goals

1.2.1 Problem Definition

Coordinating crowds. Previous methods generally utilise some form of local controller for the agents in a crowd, which plan primarily for the next state of each agent based on the current state. Such controllers are generally selfish, with each agent maximising its own return in both its local avoidance and its path planned towards the goal. For local avoidance such selfishness produces desirable realism. However, for entire crowds the lack of cooperation between agents which results from this may cause undesirable collisions and delays, especially around bottlenecks in any given scene. Current methods are very poor at avoiding such delays, especially those which will occur in some future state of the crowd. We define a function f which plans for the long term motion of the crowd such that:

$$X_{paths} = f(scene, Y_{agents}, starts, goals), \quad (1.1)$$

where $scene$ is the available space and the position and size of the obstacles within it, Y_{agents} is the list of agents to be used and $starts$ and $goals$ are the positions of the start and goal points for the agents. X_{paths} is a path for each agent for their entire progress through the scene. This f will provide more long term planning which is necessary to ensure that the crowd has the required coordination to produce dynamic motion which avoids congestion. Furthermore, we intend that these global paths would not prescribe an exact route for the agents but rather would be more akin to a series of waypoints. As a result they could be followed by any local controller, making our proposed system more of a compliment than a replacement to most current methods.

Additionally, in order for the crowd to be able to appropriately fill the scene without creating congestion at bottlenecks we also need to compute the number of agents which will fill it to capacity. Here we define the capacity of a scene as being the number of agents who can move from the given *starts* to the given *goals* without causing congestion or queueing at bottlenecks along that route (that is, filling each route up to its capacity). Thus when we talk about filling the scene we do not mean that all of the open space within the scene will be occupied, but rather that each route from the *starts* to the *goals* will be filled according to its capacity (normally determined by the width of the bottleneck on that route). Finding the capacity of the scene and suitable routes to fill it is a non-trivial problem, as there are not only a large number of ways to split up a group of agents into any given set of routes, but the interactions of those agents in the future are non-trivial to predict. To perform this task we define a further function g which provides the list Y_{agents} for f :

$$Y_{agents} = g(scene, goals), \quad (1.2)$$

here g will perform analysis of the *scene* and *goals* to create a list of agents such that they will fill the scene to its capacity, but not beyond. This prevents scenes becoming overfilled and allows f to find cooperative paths for the agents which avoid congestion.

Providing controls. Current methods, particularly agent based ones, often provide a means to control the behaviour and actions of each member of the crowd. However, there are no controls which allow a more generalised control of the crowd as a single entity. For global control of this type controls are needed which have a direct relationship to some characteristic of the crowd and the way it interacts with the scene. We propose that f could be further adapted to take input from a user defining crowd behaviours:

$$X_{paths} = f(scene, agents, starts, goals, \theta), \quad (1.3)$$

where θ is a set of controls which can be set by the user. Global planning over a large number of agents is non-trivial due to the uncertainty in how they will react to any new instructions and how those reactions will effect other aspects of the crowd behaviour. Using f and the *agents* from g will allow us abstract the crowd's interaction with a given scene in such a way that intuitive controls can be provided for control of crowds. Here we define intuitive to mean that the effect of the control can be easily understood by the user without further experimentation or explanation. In this case specifically we use coordination and congestion, thus it can be easily understood that by decreasing either of these the crowd will become less coordinated or less congested respectively.

1.2.2 Goals

The goals for this research project can be defined as follows:

1. Given a particular scene, to provide control and coordination for a crowd which will fill that scene to its capacity with a crowd which moves dynamically across the entire course of its motion. The crowd will avoid congestion and the agents will avoid crossing one another's paths such that the dynamic motion of the crowd is preserved. This will be done with minimal input from the animator.
2. Having produced a dynamic crowd scene, controls will be provided which allow alteration of the type of crowd motion which results. These controls will be simple, such that the entire crowd can be controlled with just a few variables. The controls will also have a high level of clarity, in order that the effect they will have can be intuitively understood. Finally, they will allow the creation of number of different varieties of scenes.
3. Control will be demonstrated over other types of crowd scenes, such as evacuations and team sports. This will be in such a manner that not only can examples of such scenes be produced, but that additional information is provided which gives useful tools and understanding.

1.3 Key Issues

There are several key issues and problems which need to be solved in producing the desired system:

1. **Complexity.** Planning across both the spatial and temporal space of a scene is very complex. There is an extremely large solution space within which there are very few solutions, even fewer of which are both valid and satisfy the requirements laid out in Section 1.2.2. In order to solve the problems laid out here this system must find abstractions which simplify the planning domain such that it is possible to produce suitable plans.
2. **Validity of the Planning.** The plans produced must be valid, that is they will allow agents not only to progress through the scene but to progress towards specific goals and to do so while filling the scene to its capacity along the routes towards those goals. In order to demonstrate that produced plans achieve such

validity a comparison must be provided to a previous method in the area and show that the plans represent improvement.

3. ***Dynamics of the Motion*** The routes provided by our system must create dynamic motion across the entire scene. Such motion will allow interaction between agents while still producing a high degree of motion towards the *goals* with a minimal amount of interruption or slowing. Such dynamism will be evaluated through analysing the overall motion of the crowd across the course of its progress and comparing it to another globally aware method.

Over the course of this work we will attempt to demonstrate that it deals with each of these criteria in turn.

1.4 Publications

The work presented here has been published in Barnett et al. (2013).

1.5 Methodology Overview

In order to satisfy these goals and overcome these issues a number of methods are utilised to create our system. In this section we give an overview of the methodology used in this system, with particular reference to how each method will be applied to solve the issues raised in Section 1.3.

Topological Representation Initially we compute a Harmonic field across the available space and from this we find a Reeb graph. This provides a graph which defines the topology of the scene according to the mapping of the Harmonic field. By abstracting into this topological space we can then find the Maximum flow over this graph, providing a guidance which demonstrates, in the maximum flow case, how many agents will be where in the scene. This three step process provides the solution to the complexity described in Section 1.3, allowing a generalised plan to be made which details the movement of agents throughout their progress through the scene.

Producing Routes The topological plans are converted back into the geometry of the given scene. This is done using the relationship of the Harmonic field to the Reeb graph along with careful stepping through the graph to obtain routes which are valid not only in reaching from the start to the goal, but also in their efficiency and coordination

with one another. This process helps to satisfy the validity of the planning problem described in Section 1.3.

Providing controls Next we satisfy the goal given in Section 1.2.2, to provide control over this type of crowd scene. This is done firstly by defining two variables which alter the make up of the routes provided thus far, these are called Congestion and Cooperation.

Additional Applications Additionally, the system is adapted to a couple of other problems. Firstly, to evacuation scenarios, again using the routes produced to demonstrate additional information about building structures which can be used in their design. Secondly, applying the principles of topology and maximum flow to the control of agents in team sports, allowing coordinated optimal decisions to be made for the team as whole.

1.6 Thesis Structure

The structure of the rest of the thesis is as follows. Firstly, Chapter 2 provides a review of some of the related work, then in Chapter 3 the computation of Harmonic field and associated Reeb graph are described. Chapter 4 explains how the maximum flow is calculated over this Reeb graph and Chapter 5 explains how the information in the graph is converted into paths for the agents. In Chapter 6 the experimental results of the system are presented, then in Chapter 7 we describe some additional applications of our method. Finally, conclusions are given in Chapter 8.

Chapter 2

Related Work

In this chapter we will provide an overview of the field of crowd simulation for computer graphics. There are other similar areas of study of some relevance, in robotics and the associated path planning, and in simulations for crowd study and evacuation simulation. For our purposes the work of primary importance is focussed in the area of graphics. We will be paying attention to how the works here relate to our own in regard to their relevance to our focus on coordinated crowd motion and to providing specific tools to an animator.

Within the computer graphics community there has been a significant amount of work done on the problem of crowd simulation. It is important both for easily animating scenes and for providing scenarios through which players can navigate in computer games. In both cases as computation power increases so the demand for large seamless crowds to fill in the backdrop is also increasing, as it provides a simple realistic way to provide fluid and believable settings for any scene.

Many different divisions have been proposed for the work in this field. We choose to roughly follow one of the two dimensions proposed in Zhou et al. (2010), dividing the work into **local** and **global** methods (in Section 2.1 and Section 2.2 respectively). This division was chosen as it lends itself well to comparison with our own work. Obviously there are some issues with the division, as often local systems will involve some form of global pathfinding and nearly all global works provide some local avoidance. Here we specify local control as that where the knowledge used in making the decision for each agent is primarily local to that agent's space, as in agent based methods or flocking Reynolds (1987a). For global control the focus is more on global information, often even making decisions for the crowd as a whole, as in field based methods.

2.1 Local Systems

In this section we will analyse the many different systems proposed for crowd control which are focussed on local control. Specifically, on those works which allow control of a number of agents moving at once within the same scene, as opposed to motion graphs or similar works which deal only with creating movement for a single individual. Local methods are generally preferred because they allow much more intuitive control whereby each member of the crowd is treated as its own individual agent.

Within the collection of local controllers there are a number of different approaches. Firstly, we will look at geometric based methods in Section 2.1.1, which base the rules for each agent around the position of nearby agents and a series of geometric rules relating them. Next, we will look at particle based methods in Section 2.1.2, these treat each agent in the crowd as a particle pulled about by various forces based on its local situation, its movement is then the aggregate of those forces. Next in Section 2.1.3 we will look at agent based methods which give a series of rules to each agent dictating their movements to produce a particular kind of behaviour. Finally, in Section 2.1.4 we will look at methods which directly model or create their crowd behaviour using examples from data.

2.1.1 Geometric Systems

Geometric based systems focus on the local area around each agent and the rules which it uses to avoid other agents and obstacles. As a result typically the focus is on avoiding collisions rather than on allowing control or producing realism. The advantage of this is locally smooth efficient motion, the disadvantage is that over larger time scales or distances efficiency is not considered and may be compromised.

In the field of Robotics geometric control is a much more studied area of research, though in those cases the problem being dealt with is quite different as the robots must deal with incomplete and noisy information about the world. Nonetheless some systems have crossed over from there into animation, particularly the work done on Velocity Obstacles in Fiorini and Shiller (1998). Here each obstacle is extrapolated out into the robot's velocity space to allow definition of the set of all velocities which will cause collisions, this allows the choice to be made from the remaining velocities which will avoid collisions. This allows a single representation to take account of the collisions with all nearby obstacles, the disadvantage is that searching the reduced velocity space for an appropriate velocity is quite inefficient.

The main extension of the Velocity Obstacle method, which has been applied in Robotics but also in Graphics, is the reciprocal velocity obstacles (RVO) method presented in van den Berg et al. (2008a). Here the other agent obstacles are assumed to be using a similar avoidance methodology which will result in them also reacting to the current agent, this leads to reduced avoidance. The representation of obstacles in general is also improved by representing each one by a much simpler half plane on the velocity space, this reduces the search for a new velocity to a set of linear constraints defining a convex region, massively simplifying the search. Even so there are still problems whereby agents can fail to pass one another causing oscillations.

These problems of oscillations in RVO have been tackled in a number of ways. In Snape et al. (2011) it is enforced that the agents pass one another on a particular side by enlarging the velocity obstacle on that side. Equally in van den Berg et al. (2008b) the system is extended with simple global plans for each agent computed at every step. Nonetheless both of these approaches still suffer from poor foresight which can result in extreme congestion and even deadlock at bottlenecks.

There are other approaches which deal with local geometric details outside of the RVO framework. In Guy et al. (2009) the velocity obstacles method is parallelised on a number of levels with some additional relaxation of the collision constraints. However, the changes from the original velocity obstacles are few enough that it has many of the same problems with oscillations occurring. The work in Jund et al. (2012) offers a simplification of the process of checking for other agents to avoid by dividing the space into a variable resolution mesh, allowing checks against only significant regions of the mesh. In Kapadia et al. (2009) each agent projects an egocentric field across its local path. This field allows for local path planning, but also for avoidance between agents as they use the field to detect areas into which other agents intend to travel and treat them as less desirable for pathfinding as a result. Similarly, in Golas et al. (2013) agents estimate the potential for future collisions by representing other agent's future position with Gaussians which are increasingly less well defined the further ahead they are being projected. This allows avoidance to be carried out which lets agents to see the potential obstacles presented by large crowds, without being unduly effected by individual agents. In Singh et al. (2011) agents use simple space time prediction to estimate where other nearby agents will be, they also use reactive motions to take account of unexpected near collisions, all of which is done with each agent having a limited (and so more realistic) perceptual field.

The work in Kapadia et al. (2009) and Singh et al. (2011) are compared to one

another and to an implementation of the RVO method in Kapadia et al. (2011a). Here they analyse how close the methods come to the optimal path length and how close they come to the optimal time, finding roughly that they achieve about 75% of optimal performance. More significant for our purposes is that they also found that all three methods were only able to successfully handle 60% of the chosen scenarios, where failure is characterised by either a significant collision or a complete deadlock. The fact that there were so many failures with all of these geometric methods tested seems to indicate that they are not, on their own, enough for a full crowd control solution.

2.1.2 Particle Based Systems

Particle based methods treat each member of a crowd as an individual particle which is influenced by a series of forces, its movement is found by taking a weighted combination of these forces. In this way various behaviours such as avoidance and leader following can be combined into a single crowd behaviour model. This is the oldest form of crowd simulation and it was initially introduced in Reynolds (1987b). Here the local forces combine to form a crowd which moves in a way reminiscent of flocks of birds. To produce the forces required to do this each agent looks at other local agents and produces three forces, one to make it move to avoid any others which are too close, one takes their average direction of travel and moves to match it and one finds their average position and moves to match it. The produced flocking behaviour is an emergent result of these three simple forces.

The issue with the emergent flocking behaviour created in Reynolds (1987b) is the very fact that it is emergent from the interactions between three forces. As a result it is difficult to predict what effect additional forces will have on the behaviour created and so it is hard to put the method to specific use. Some attempt was made in Bayazit (2004) by making use of a Probabilistic Roadmap of the environment to allow the flocking agents to explore the environment or steer towards specific locations. However, extending such particle based systems has been abandoned recently by the graphics community in favour of more modern methods.

One area where particle based methods are still widely used is in evacuation simulation, for the purpose of designing and evaluating real world scenarios and buildings. This is generally done through the social forces model presented in Helbing and Molnár (1995). Here again there are three forces involved, an avoidance force which attempts to maintain a certain distance from other agents and obstacles, an acceleration

term which attempts to move towards a desired goal at a desired velocity and a third term which models attractions between agents, allowing for, for instance, the creation of small subgroups within the crowd representing families moving together. The big differences with these forces is that they are produced from non-linear equations and that the values which they use (such as the desired velocity) are in most cases taken from actual data gathered in real crowds. Due to the increased realism of this representation additional behaviours can be created in a more intuitive way, for instance in Helbing et al. (2000) it is described how panicked crowds (and their associated problems with crowding and congestion) can be modelled in part by increasing the desired velocity of the panicking agents.

The above methods have been partially combined for a more recent work in Pelechano et al. (2007). Here there are a large number of local forces modelled on actual real world data as in Helbing and Molnár (1995), however, the equations for their computation are mostly either geometric or linear as in Reynolds (1987b). This method allows modelling of a number of crowd behaviours, from natural lane formation when groups of agents pass one another, to panic and its propagation through the crowd, at a much faster speed than the full social forces model.

Particle based methods provide an emergent mimicry of human behaviour, but this means that they are only reactive and allow little space for user control. The emergent nature of the combination of different forces also makes adding extra behaviours an unpredictable process.

2.1.3 Agent Based Systems

Agent based methods are those where each agent is controlled by its own individual set of behaviours and rules and they are similar to particle based methods in many ways. The primary difference with agent based systems is that these behaviours are mutually exclusive states, with only one active at any time. This makes it considerably simpler to add new behaviours for agents, as they will be active only on their own so there is much less chance of them interacting unexpectedly with current behaviours. The classic example of this is Tu and Terzopoulos (1994), here the agents controlled are fish with behaviours deciding whether they are predators or prey and whether they are schooling fish. The decision about which behaviour a given agent should be following at any time is based on a decision map, which is navigated based on the senses of the agent (such as whether the fish is about to collide with anything). The chosen

behaviour is then followed using local geometric controllers until the decision map is rechecked at the next time step.

An early example of agent based control for crowds can be found in the work in Musse and Thalmann (1997) and Musse and Thalmann (2001). Here the control of large crowds is simplified by reducing them to larger groups of agents which can be controlled in a flocking type method. The grouping is a state which agents move into and out of. Each agent has a list of goals and interests and they choose to move into or out of a group based on a comparison of these interests. This method is extended in Musse and Thalmann (2001) by adding a script based language which allows simple IF and WHEN logical statements to describe additional environmental or conditional behaviours.

The work from Tu and Terzopoulos (1994) is extended in Shao and Terzopoulos (2005) for use in pedestrian crowds with a focus on populating historical scenes, such as an ancient temple in Shao and Terzopoulos (2006). Here decision networks are used for much the same purpose as the logical statements described above, to decide upon specific higher level behaviours based on the current situation, such as moving to a certain shop in the scene. This work in Yu and Terzopoulos (2007) provides a natural extension to this, with more complex decision networks which allow the representation of decisions involving more than one character and reactive agents who take into account the behaviour and state of surrounding agents. The lower level avoidance behaviours throughout these methods are handled by checking for particular pedestrian states, such as an oncoming collision or finding oneself in a temporary crowd, these situations are checked for one by one, with each activated behaviour overriding the previous ones. The ordering for the checking of these avoidance behaviours was decided by exhaustively checking all possible permutations to find the one which caused the least collisions, this fact means that unfortunately adding in new such behaviours would be a time consuming process.

A classic approach in agent based control is the use of a Finite State Machine (FSM) to determine the movement of agents between different behaviours and an example of this can be seen in Sung et al. (2004). Here the current state of the agent provides a probability which determines how likely it is to move between states in the FSM, such as moving from heading to a target to performing local avoidance. They also allow users to define location specific extra behaviours for certain areas of a scene, such as sitting on a bench, and these can easily be added as simply an extra state in the FSM.

It is also possible to describe the agents at a much higher level, where the different states represent different psychological states or personalities. For instance in Guy et al. (2011) they demonstrate how to create agents who are shy, tense or have a high level of extraversion, all based on results from personality theory. These are represented using the RVO approach from van den Berg et al. (2008a), with different personalities emerging from the use of different parameters, for instance extraverted agents have a larger radius and prefer to move faster. In this method the agents cannot move between different personality states, but rather they allow the creation of heterogeneous crowds where agents can be seen to have a variety of reactions to various scenarios. This work was extended in Kim et al. (2012) where the extraverted and introverted personalities instead represent two different behaviours for the agents. Agents in this work are typically introverted and careful, but when their personal stress value reaches a certain level (caused by situations such as a nearby fire) then they switch to being extroverted and become more aggressive in pursuing their goals. This allows the simulation of scenes such as evacuations, where panic slowly spreads through a crowd. In Yeh et al. (2008) agents with different personalities are again represented, this time their different personalities are created through the addition of composite agents. Composite agents are additional hidden agents who only appear in the avoidance portion of the simulation making them effectively an area of repulsive force, this allows them to clear areas such as doorways of other agents without any actual obstruction. These composite agents are used to simulate personalities by attaching them to other agents, for instance creating a more aggressive agent by attaching a composite agent ahead of them, simulating the way they force their way through a crowd and push other agents out of their way.

Agent based methods are extremely adaptable and can produce a wide variety of different and heterogeneous results. However, though they have many different applications, their primary problem is that they must be hand crafted towards whatever task they are required for, and the agents created for one scene may not be suitable for another, requiring a user to create a whole new agent.

2.1.4 Crowd Control from Data

Crowd simulation from data involves recording a crowd at some level of detail, then taking the recorded crowd and attempting to replicate it. The idea is simple, that by modelling based on actual crowds the realism of movements will be guaranteed. There

are a great number of approaches taken to this problem, and some of them even allow control of particularly large crowds of agents, but the focus is still always on the local interactions between agents within those large crowds.

One class of approaches involves trying to replicate local behaviour of individual pedestrians. A particularly low level example can be seen in Arechavaleta et al. (2006) where people were tracked as they performed local path planning such as moving so they are in a certain spot facing a certain direction. The results show that they follow a series of clothoid arcs (curves whose curvature grows as they move from their origin), this is extended to a full control method in Laumond et al. (2011), where simulated human paths are described through increasing or decreasing the curvature over the course of the motion. Such highly detailed methods have been used for graphical simulation in Pettré et al. (2009), here the focus was on simulating scenes of people passing one another. Observing such motion allows them to create geometric rules which describe aspects of the motion, such as the fact that the avoidance is not merely reactive but also includes an observation phase. The produced motion allows avoidance with minimal disturbance from their original path for both agents, though they admit that such considered avoidance may not be valid for more crowded situations. Similar principles are used for a full crowd simulation method in Guy et al. (2010), here the method is based on replicating the Principle of Least Effort, an observation that organisms of all kinds will always attempt to expend the least effort possible to reach their goal. This motion is created by optimisation for each agent which attempts to minimise the time it takes to reach the goal while staying at a constant velocity. In their method of optimising the velocity there are some similarities to RVO, as a similar velocity space representation is used. Again they demonstrate that when simulating two agents crossing the avoidance motion involves only a minimal disturbance from the agents original motion.

Another set of approaches involve trying to match the current crowd situation with observed real world ones, the observed movements can then be replicated. This approach can be seen in Lerner et al. (2007), here captured data was described in terms of the awareness of an active agent and their current situation. Agents make decisions by finding the captured situation which best matches their current awareness and acting as in the same way as in the data. This method was extended in Lerner et al. (2009) where the data was manually labelled based on the action it represented. This allows the actions to be grouped by type into an FSM and the comparison is instead made to each entire group of actions improving the accuracy. Another approach is found in Lee et al. (2007), here again the data is captured of crowds from above, but the data is not

labelled, instead linear regression is used by the agents to match their current situation to the closest one from the data. This control is paired with a high level decision mechanism, which allows agent to deviate from the data. Finally in Ju et al. (2010) a number of crowd formations are taken as input, some from data and some artificial, with each formation divided into distributions of similar situations which represent it. The agents choose a trajectory from the set of all available ones used in their given formation, then the result of that trajectory is followed if it results in a formation which matches one of the associated distributions. The method also allows blending between two crowds using bipartite matching to match sets of points in the distributions in either formation and then linearly blending between the two to produce a set of distributions for a blended formation.

An approach which does not use captured data is presented in Yersin et al. (2009). Here the data used is made up of a number of patches, each of which is a cyclical 3 dimensional representation of the motion of agents through a crowd, with one of the dimensions being time. By laying down these patches one at a time each new patch can be made to match the patches on either side, with agents entering it where they leave them and vice versa. Matching up entry and leaving points, performing smoothing and avoidance, and applying some restrictions (such as ensuring that agents cannot move back through time) allows the creation of a valid new patch. A scene can then be made up entirely by adding patches one at a time. This is an interesting method, but it has a few drawbacks, firstly it is not clear how well these patches would operate in a very crowded scene, as the avoidance internal to each patch is a simple forced based method, secondly as each patch is cyclical a user looking at one point for too long might be able to notice it repeating itself.

The final type of approach to creating crowds from data involves recording the motion of crowds or groups of agents and using it with as few changes as possible to preserve the interactions. One example of this is in Kwon et al. (2008), here the motion of a group of agents moving through a scene is represented as a mesh where nodes are the positions of each agent at a particular time step and edges represent either connections between those agents or connect the agents to themselves at time steps before and after. By using Laplacian mesh deformation techniques the motion can then be twisted to allow a motion from one scene to fit into another differently shaped scene, with some small scale corrections to ensure no agent is forced to move too fast by the deformation. These ideas are extended in Kwon et al. (2008) to allow the deformation and editing of actions where agents are directly interacting (rather than just walking

along near one another), with some additional restrictions to ensure that the point of interaction is properly preserved. In Lai et al. (2005) a method is introduced that attempts to replicate the motion graphs method from Kovar et al. (2002) for crowds. The entire motion of a group of agents is broken into similar chunks, these are then connected up into a FSM such that new motion can be created by moving between the states and assigning agents positions from the chunks associated with each state. These positions are then followed using flocking, with matching the position within the chunk as one of the agent's forces. Finally, large crowd scenes are constructed using much smaller pieces of captured motion in Shum et al. (2008). Here pieces of motion which allow valid interactions between multiple characters are combined into interaction patches, these interaction patches are then combined with one another to produce much larger motion comprising many agents across an entire scene.

As this demonstrates, there are many useful methods which allow the creation of crowds from data. The primary problem with this approach is that actually obtaining and processing data of large crowds is extremely difficult and costly. As a result using these methods is, for the moment, restricted to local simulation.

2.2 Global Systems

In this section, we will analyse the various methods for crowd simulation which are based on global control. These are distinct from local controllers in the fact that they primarily take into account global factors of the scene. They normally also include local controllers, for collision avoidance and other finer adjustments, but the focus is on the global control provided. Using such global information allows the crowd to be more coordinated and more easily controlled, but this generally comes at a cost to the realism of the crowd, as individual members of a crowd generally don't have global information. The methods in this section provide a much more relevant comparison to our own proposed method, as our method is also primarily global in its solution to the crowd control problem, as a result we will attempt to give much closer comparisons to our system and the reasons for the differing choices which we made.

There are a number of different approaches to global control and for the purposes of clarity we have divided them into subsections based on the types of approaches which they favour. The first of these are field based methods, described in Section 2.2.1. These utilise some form of field computed over the free space of the scene. Whether this field accounts for other agents and how it deals with obstacles varies, but in all

cases the field is used to direct the navigation of the agents. The other type of approach is topological based methods, described in Section 2.2.2. This is a much broader banner but it serves to describe those systems which utilise topological methods in controlling the agents. In some cases this is used to produce a simpler description of the scene over which more complex routes can be planned, or in other approaches topological approaches are utilised in the production of the scene in more oblique ways. As with any division these categories are not perfect, in particular many topology based systems utilise fields in some form, but the papers have been divided according to what we saw as their primary focus.

2.2.1 Field Based Systems

Field based methods involve computing a field across the space, taking into account the obstacles along with other factors, and then using this field for navigation. They may be continuous, but generally they are approximated on a grid of the free space for ease of computation. These types of approaches have been around for a long time in other areas of research, such as potential field based methods in robotics as in Krogh (1984), and fluid simulation methods used in crowd simulation for the study of evacuation and building design such as in Henderson (1974). It is only more recently that some of these methods have percolated into the arena of computer graphics.

One particular type of field used in crowd control is the Harmonic field. They are useful primarily due to their lack of local minima and in robotics they were first introduced in Connolly et al. (1990). An example of this type of field control is Silveira et al. (2010a) and it was later applied for gaming in Silveira et al. (2010b). Here a discretized Harmonic field is generated by setting the potential value of the edge of every obstacle and the arena to be 1 and the potential value of the target to be 0. This ensures that wherever an agent is in the scene they can always follow the gradient of the field to arrive to the target. However, because every obstacle has such high potential it also means that agents will always follow the centre of any corridor rather than being able to spread out through the space, leading to much less efficient routes being used and groups of agents crowding together.

Another approach is to use the geodesic distance field to direct agents towards the goal. This is done in Torchelsen et al. (2010) for arbitrary surfaces. Here they compute the geodesic distance in a multi-resolution manner, using a fast and rough calculation to immediately direct agents once they get a new goal, then refining this direction in

later iterations as a more detailed calculation is made. The local avoidance is then carried out using the GPU. The basic method used here is quite well known, its main advantage is its ability to work on arbitrary surfaces, though that isn't an especially desirable quality. Another example which uses a geodesic field is Patil et al. (2011). Here a simple gradient field is computed using Dijkstra's algorithm to compute the distance to the goal from each point, and so the direction that agents should be moving from each grid cell. The big difference is that they allow the user to draw directed lines onto the space which agents should follow. These directed lines are taken as the gradient in the cells which they cover, then for adjacent cells the gradient is taken as an average of their surrounding cells to ensure smooth transitions between the directed cells and the rest of the gradient field. The result is a simple field which directs agents towards the goal, but through which users can define the specific paths or routes which they want agents to take. This is a powerful method and it allows a level of control which ours does not, but we would argue that for large scale examples, with many obstacles, it would be infeasible to define the direction for agents along every corridor (a process which our system takes care of automatically).

The final and most influential form of field based crowd simulation is based on fluid dynamics. As previously mentioned in Henderson (1974) in the field of engineering and evacuation simulation, fluid dynamics have long been used. In Narain et al. (2009) they follow the fluid simulation quite closely. Here each grid square of the space computes the average flow through that square, based on the desired movement of each agent towards its goal. The flow of adjacent squares are then combined, taking into account the universal incompressibility of fluids, this provides a new flow of each square which prevents it from crushing nearby squares. The agent's final motion is then a combination of its desired motion and the final flow found from the square. Using incompressibility like this ensures that extremely dense scenes can be simulated without collisions between agents, something which is not possible in any other method. In situations where the incompressibility does not come into account though (with less dense crowds) the method approaches a grid based version of flocking.

More recently in evacuation research inspiration was taken from fluid based methods, but instead the crowd were treated as so called thinking fluids, that is, a continuum whose behaviour is effected by its situation, this method is summarised in Hughes (2003). Here the crowd is treated as a continuous flow, but there are three additional assumptions used in the equations for its control. Firstly, that an agent's speed is determined by the density of the crowd, secondly, that agents try to reduce their distance to

the goal and finally, that agents attempt to avoid areas of high density. A potential field is then computed taking into account all of this, according to the agent's current knowledge. This work has been applied across a range of simulations including simulating the flow of pedestrians in Mecca Hughes (2002) and simulating ancient battle scenes in Clements and Hughes (2004), demonstrating its efficacy as a simulation methodology. This approach was adapted to the field of graphics research in Treuille et al. (2006). Here they follow closely the assumptions from Hughes (2003) but discretising the computation across a grid and using the additional assumption that each agent has global knowledge about the scene. This assumption allows agents to re-plan to avoid congestion at a bottleneck even if they cannot currently see that bottleneck. This is useful, but it also slows the method and in order to achieve a good enough performance they had to simplify the approach by separating the crowd out into groups and treating each of them as a single agent for the purposes of computing the potential field (as otherwise each individual agent would require its own field calculation). This work is extended somewhat in Jiang et al. (2010) where they allow computation of the field in complex environments. This involves altering the calculations per grid such that, in areas with finer detail, a lower grid size is used. They also demonstrate how multilevel scenes are connected for calculating the field up stairs and add a discomfort field to obstacles to improve the method's performance around bottlenecks. The Continuum method for is clearly useful for graphics, it can deal with controlling very large crowds and provides realistic results. Still, though it allows the agents full knowledge of the scene, there is very little planning ahead. There is some, in the form of a discomfort field which each agent projects out in front of it, this means that when a bottleneck is about to become full agents will avoid it, but the discomfort is not projected far enough to allow it to provide much predictive power. As we plan over the full duration of any scene, avoiding congestion at such bottlenecks is the primary advantage which our method has over the continuum approach.

2.2.2 Topology Based Systems

Topology based methods deconstruct the scene or the crowd into a series of simpler representations which describe its shape. This abstracts away all but the detail which is necessary for pathfinding and allows much more advanced planning across the space for the motion of individual agents or a whole crowd. Equally, the representation may be such that it allows for much simpler search based planning methods, such as A*

search, to explore through it and find plans much more easily than they could on a simple grid. In this section we will review the methods which create these topologies and the various different ways in which they are used.

Probably the classic topological method in graphics is the navigation mesh, Snook (2000). This involves dividing the space up into a mesh made up of a number of distinct regions. The navigation can then be done based on the connectivity of those regions to one another and further decisions can be made based on the situation within those regions. There is no commonly defined method for calculating such a mesh and a number of different approaches have been proposed. In system in Pettré et al. (2006), Morini et al. (2007) and Maïm et al. (2009) they follow a method in which the entire space is divided up into navigable cylinders. This is done by first taking a random sampling of points in the space, these points are filtered to ensure that they are evenly spaced and that those furthest from the edges are used. Cylinders are then placed over each of these points with a radius equal to the distance to the edge of the navigable area at that point. Finally the edges of the mesh are found by looking for any intersection between two cylinders. This intersection also provides an line defining where the two areas meet and this allows control of where agents cross between regions. Finally the path planning based on this method is done using Dijkstra's algorithm to make an ordered list of the paths through the space, and agents decide on which path to use based on the density of agents along it, with potential fields used to allow local obstacle avoidance.

In the method in Geraerts (2010), for a set of convex obstacles the space is divided up into Voronoi regions, those areas which are closer to one obstacle than any other, which separate the space into a set of walk-able areas. Also calculated is the medial axis, which is a group of lines describing all the points equidistant from more than one point on the obstacles, these describe connections between all of the Voronoi regions which are guaranteed to have the highest clearance. Routes can then be described in terms of the medial axis giving a corridor of allowable movement. As the distance to nearby objects is known at all points along these routes this allows plans to be made which can account either for the width of agents or which give the maximum clearance from all obstacles. This approach is extended in van Toll et al. (2012b) where path planning is done more directly over the medial axis using A* search, with additional re-planning to find new paths in cases when density is too high. However, unlike in Pettré et al. (2006), here the density replanning only takes account of high density areas which are near the agent, as more distant sections of the path may become

less congested long before the agent arrives there. Equally, the corridor map method from Geraerts (2010) is utilised in Karamouzas et al. (2009), which allows users to draw a route through the space which they want followed. By matching this route to the set of Voronoi regions which it passes through, the medial axis which corresponds to the route can be found. This medial axis path provides a corridor and itself is a maximum clearance path, both of which are used along with the originally entered route in a force based method to pull agents along the route. Finally in van Toll et al. (2012a) they explain how the kind of mesh used in these papers can be recalculated on the fly to allow the mesh to account for dynamic obstacles. This is done by finding the set of local points based on the Voronoi regions which have been interrupted by the new or moved obstacle. The Voronoi regions can then be updated by iteratively re-examining the relationship of them to only those local points.

Graph based methods often create their graph in much the same way as mesh based methods, but here the focus is on graph itself with the regions represented by any individual node ignored. The classic example of a graph used for navigation is the Probabilistic Roadmap method described in Overmars (1992). Here points are placed at random throughout the open space of a scene, with connections being drawn between close by pairs based normally on local path planners. However, the graphs which are produced as a result generally end up being overcomplicated, with many unnecessary loops and dead ends which make pathfinding and planning difficult. As a result of this complexity most recent methods do not use graphs except where they allow the application of graph theory methods from mathematics. One such example is Lamarche and Donikian (2004) where the graph of the space is built by performing Delaunay triangulation on the obstacles and boundary of the arena. A set of edges is added to this describing the shortest distance between each corner and the nearest wall. Finally the produced mesh is simplified by merging adjacent triangles to produce a series of convex cells, doing so while still preserving local bottleneck information. The resulting set of cells can then be connected up to produce a graph describing the topology of the space, this graph is then further simplified by removing dead ends and collapsing corridors into a single node, while preserving the topology. This allows very quick path planning on the simplified topological graph which can then be converted to a more complex plan in the actual space. Local pathfinding and avoidance is then carried out using a series of pre-defined rules. Due to the topological abstraction used, it is not clear whether agents take the length of their chosen path into consideration and unlike previous methods they do not consider the density of the path to be followed

So far three topological methods have been presented and though on the surface they are approaching the problem of controlling a crowd in a similar way to our system, in practice we believe there is a single significant difference. Namely that although all of them allow agents to use global information when making their choices, they are doing so individually with no coordination between the different members of the crowd. It is our belief that the ability to produce such globally coordinated control is one of the advantages of using a topological abstraction. One example of this can be found in van den Akker et al. (2010). Here the scene is represented as a graph with directed edges each of which has an associated traversal time. They then model the path planning problem for a crowd at one point on the graph as a dynamic flow problem, which attempts to compute paths for all the agents such that they all arrive at the goal in the shortest possible time. Dynamic flow is an NP hard problem, so they solve it iteratively with column generation, starting with a simple path and adding new ones only when they will reduce the cost of getting the agents to the goal. This work is further extended in Karamouzas et al. (2013) which allows undirected edges and takes into consideration the capacity of nodes in the graph. It also explains the generation of the graph and providing of paths to the agents using the medial axis as presented in Reynolds (1987b). These two papers solve a very similar problem to our system and are capable of solving certain problems which we cannot, such as non-homogeneous agents or non-static scenes. However, their approach is somewhat different as they deal with moving pre-existing agents between locations rather than filling a scene for an animator. They also offer no solution to the problem of crossovers which can occur when several agents following a graph move into and out of a single node in intersecting paths, we discuss this issue, along with our solution, in more detail in Section 5.1.

Chapter 3

Harmonic Fields and Topology

Discovering the topology of a scene is central to the effectiveness of our method. Given a scene with as few as 2 obstacles forming a bottleneck, there may be points of congestion which are very difficult, if not impossible, to plan for with either agent based or field based methods of crowd control (examples of this can be seen in Section 6.3). The topological map provides full information which demonstrates how the available space can be navigated and using this it is possible to anticipate obstructions and navigations well ahead of time.

The topology is found using a Reeb graph computed over a Harmonic field. There are a number of other approaches which would also yield the topology of the scene, in particular the medial axis has proven popular. The reason we use the Reeb graph is because of its strong links to the Harmonic field, this has a definite gradient from the start points to the goal points which are used throughout the system. Specifically, they are useful to provide paths for the agents, as described in Section 5.3, they also allow the optimisation of the maximum flow calculation by using them to fill in more desirable routes first, as described in Section 4.3, finally and most importantly they allow the routes through the space to be put in order after computing the maximum flow to allow the system to avoid giving agents paths which cross one another, as described in Section 5.1. In addition by specifically calculating the Harmonic field based on the position of some start and goal points this considerably simplifies the structure of the produced Reeb graph, making the computation of the maximum flow much easier.

In this section we will first explain why the Harmonic Fields were chosen to represent the space and then how they are computed. Next, we will talk about how they are used as input to the Reeb Graph method to produce the topology of the scene. Finally,

we will cover the method used to compute the flowlines with some details about the simplifications which we found were necessary and the justification for their use.

3.1 Harmonic Field

In this section we will give the details regarding our use of the Harmonic Field. First, we will provide the decision process which led to its use as the particular input for the Reeb Graph. Next, we will also describe the precise method for its calculation. Next, the method for calculating the gradient of the field will be explained. Finally, the computation of the paths through the space generated from the field, called Flowlines, is explained.

3.1.1 Use of the Harmonic field

The Reeb Graph can take as input any continuous scalar function. The only requirement we had for the system was that we wanted to compute the crowd movement plans for a given set of start points and end points. With this in mind a measure of geometric distance between these points originally seemed like a good fit. However, early experiments showed that when following the gradient this only provided the shortest path between these points. This is a problem as we wanted to use the gradient of the function to provide paths for our agents to follow, and this is what led to the decision to use the Harmonic field.

The Harmonic field is computed with the only extrema located at the given start and end points. This means that there are no local extrema and the Reeb Graph will begin and terminate precisely on those points, simplifying the path planning problem considerably. By setting its values to zero at the start points and one at the end points it also has the highly desirable property that following its gradient upwards is guaranteed to provide a path towards an end point. From this it is reasonable to assume that this gradient provides a good approximation of the direction which each agent will be moving in at all points in the available scene. This assumption allows us to use the Harmonic Field for other purposes such as providing paths for the agents to follow, as explained in Section 5.3, and gauging the available width of the space as agents move through it, as explained in Section 4.1.

The harmonic field copes easily with non-convex obstacles (an example of this can be seen in Section 6.6.1 in Figure 6.20, where the non-convex obstacle is made up

of several overlapping convex obstacles). It can also be adapted to cope with non-axis aligned obstacles and irregular polygons as demonstrated in Dong et al. (2005). However, for the purposes of our work here we stick to a grid based representation.

3.1.2 Computing the Harmonic field

The harmonic field is computed by discovering a discrete harmonic function over a computed mesh applied to the space itself. This mesh takes the form of a series of grid squares of n length, each bisected into two equivalent triangular polygons. Also applied to this mesh are the start and goal points, whose values are constrained to 0 for start points and 1 for goal points.

The method for computing the harmonic field is the method presented in Dong et al. (2005) however, we are able to simplify the weight calculations involved due to the regular grid which we use to represent the space. In this paper they use the harmonic field as a method to compute smooth edges across three dimensional objects, for the purposes of quadrilateral remeshing.

For the use of this method the harmonic field values are computed for each value of the grid, represented by u , such that:

$$\Delta u = 0, \quad (3.1)$$

where Δ is the Laplace-Beltrami operator. In order for this to be computed weights for the edges of the grid must be set, this is again done using the methods from Dong et al. (2005), as shown here:

$$w_{ij} = \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij}), \quad (3.2)$$

where w_{ij} is the weight of the edge between the vertex i and j . α and β are the two angles opposite the current edge within the two triangles which it helps define, as shown in Figure 3.1.

These weights have an added advantage as they allow us to represent different kinds of spaces. They can easily adapt to different types of grids, especially using this calculation. They could also be altered to represent the difficulties with travelling through variations in terrain, such as up a hill, down into a valley or across swampy ground.

With the weights calculated it is possible to rewrite equation (3.1) into the solvable linear system:

$$\mathbf{A}\mathbf{u} = \mathbf{b}, \quad (3.3)$$

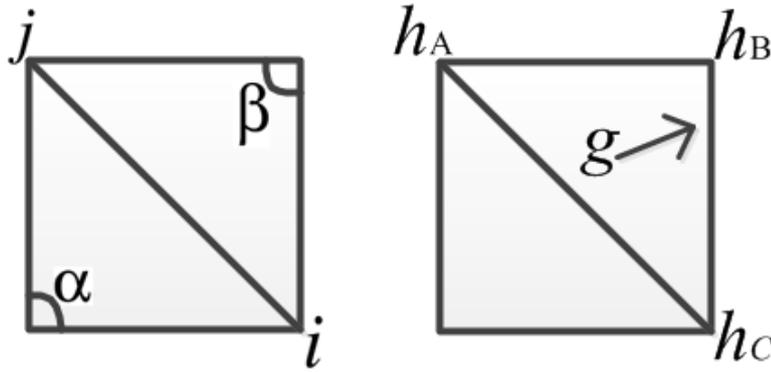


Figure 3.1: The left image shows the angles α and β used to compute the weight of the connection between i and j as described in equation (3.2). The right image shows the harmonic field values h_{A-C} used to calculate the gradient g as described in equation (3.6).

where:

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } i = j \\ w_{ij} & \text{if there is an edge from } i \text{ to } j \text{ and } i \notin C \\ 0 & \text{otherwise} \end{cases} \quad (3.4)$$

and

$$\mathbf{b}_i = \begin{cases} c_i & \text{if } i \in C \\ 0 & \text{otherwise} \end{cases} \quad (3.5)$$

and C is the list of constrained values c_i for the nodes i which are either start or goal points (their constrained values being either 0 or 1 respectively) and the weights denote the edges within A . This problem now becomes a system of sparse linear problems and as such is suitable for solving with sparse linear solvers specifically aimed at this kind of problem, in this case CSparse, Davis (2006), is used.

These results can be easily plotted to demonstrate and some examples are shown in Figure 3.2. Here the smooth shape of the field can be seen in image (a). A scene with obstacles is seen in image (b), the influence of these obstacles on the field is evident. The obstacles are included into the computation by simply removing the grid where they exist (or, in practice, choosing to ignore these connections), thus forcing the computed field to compensate appropriately. Also shown in (c) is an example with multiple start and end points. This extension takes no extra computation and it is easily set up by the user. In fact entire areas within the grid can be set to either extreme to produce a variety of effects. However, as described in Section 5.2.3 this cannot be used to create crossing flows, as a result it can only produce a subset of the examples

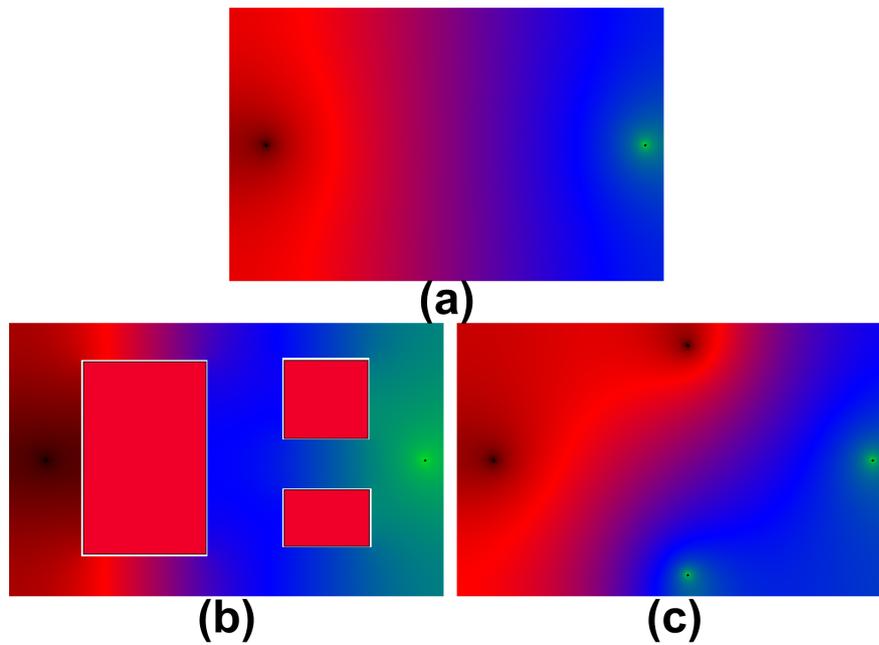


Figure 3.2: Three example Harmonic fields. In (a) there is a start point on the left and a goal point on the right, (b) has the same configuration of start and goal but with three obstacles included and (c) demonstrates the effects when there are multiple start points (on the left and top) and end points (on the right and bottom).

with multiple start and end points where there is no crossing. As discussed in Ni et al. (2004), even in the case that there are multiple start or end points there still will not be any local minima in the field. However, there will be saddle points, flat areas in the gradient of the harmonic field. If agents were sent directly along the gradient of the harmonic field this could cause them to become trapped. However, because we compute a series of flowlines as paths for the agents paths which become trapped in a saddle point are discounted before being given to an agent.

3.1.3 Finding the Harmonic field Gradient

As explained previously in Section 3.1.1 the harmonic field is also used to discover paths for the agents to follow and to determine the width of those paths in the direction perpendicular to the one we expect them to be following, the methods for finding each of these can be found in Section 3.1.4 and Section 4.1 respectively. In order for either of them to work we need to first discover the gradient of the harmonic field in each polygon.

The methodology used for this is similar to that applied in Dong et al. (2005).

However, as we are only operating in a plane (unlike over three dimensional objects) we do not have to take account of a third dimension in our equations. As a result we can simplify their gradient equation to get:

$$\begin{bmatrix} \mathbf{v}_A - \mathbf{v}_B \\ \mathbf{v}_C - \mathbf{v}_A \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \end{bmatrix} = \begin{bmatrix} h_A - h_B \\ h_C - h_A \end{bmatrix}, \quad (3.6)$$

where A , B and C are the three nodes of the polygon, \mathbf{v}_A is the vector position of the A node and h_A is the harmonic field value at the A node, as shown in Figure 3.1. The only unknowns in this equation are g_0 and g_1 , which are the gradient of this polygon in the x and y direction respectively. As a result it is possible to solve this equation by shifting the matrix of vectors over to the right hand side resulting in the following solution:

$$\begin{aligned} g_0 &= \frac{(h_A - h_B)(\mathbf{v}_C \cdot y - \mathbf{v}_A \cdot y)}{\det} + \frac{(h_C - h_A)(\mathbf{v}_B \cdot y - \mathbf{v}_A \cdot y)}{\det} \\ g_1 &= \frac{(h_A - h_B)(\mathbf{v}_A \cdot x - \mathbf{v}_C \cdot x)}{\det} + \frac{(h_C - h_A)(\mathbf{v}_A \cdot x - \mathbf{v}_B \cdot x)}{\det} \end{aligned} \quad (3.7)$$

where $\mathbf{v}_C \cdot y$ is the y component of the vector \mathbf{v}_C and \det is the determinant of the vector matrix, that is:

$$\det = (\mathbf{v}_A \cdot x - \mathbf{v}_B \cdot x)(\mathbf{v}_C \cdot y - \mathbf{v}_A \cdot y) - (\mathbf{v}_A \cdot y - \mathbf{v}_B \cdot y)(\mathbf{v}_C \cdot x - \mathbf{v}_A \cdot x). \quad (3.8)$$

The gradients for each polygon can now be computed, after which they are normalised. This provides a definition of the shape of the space according to the harmonic field at every localised spot which, as previously mentioned, will be used throughout the rest of the system.

3.1.4 Flowlines

In this section we explain how the gradients computed in the previous section are used to compute flowlines across the space. These flowlines follow the gradient of the harmonic field from 0 at the start points to 1 at the end points. They are useful for a number of purposes which are explained in more detail in Section 4.3 and Section 5.3.

For the moment it is only necessary to say that their importance lies in the fact that we assume that they are representative of the paths which agents will take through the space, as they follow the gradient, leaving from a start point in any direction will always (with a few small exceptions) take an agent to the goal.

In Dong et al. (2005) they use a method of producing flowlines which carefully tracks the gradient of the harmonic field, ensuring that the produced line enters and

leaves each polygon at the correct place along each of its edges. This leads to a number of special cases which must be detected and dealt with (such as when the gradient converges on the edge between two polygons or when the line being tracked lands precisely on the node of a polygon). Our experimentation with this method showed that the increased complexity made it much slower to trace each flowline and that the increased accuracy made negligible difference to the overall shape of the resulting flowline.

As a result of the complexity of this method we used a faster, simplified method to trace out flowlines in the space. The small loss of accuracy is acceptable for our purposes as we are providing general paths rather than exactly re-meshing objects as in Dong et al. (2005) and the inaccuracy only occurs at the polygonal level. Here we provide a definition of that simplified method. Firstly, each flowline is seeded by picking a point anywhere in the space. Two separate flowlines are now built from this point, one moving down the gradient back until it reaches a start point and the other moving up it until it reaches an end point. These two flowlines can then be joined (after first reversing the initially generated one) to provide a full route from the start to the goal point.

The same process is followed in building a flowline forwards or backwards (though in the backwards case the negative of the gradient is used). The gradient is found for the current point and then the next point is placed in the direction of the gradient at a distance of half the width of one edge of the mesh defined in Section 3.1.2. After this move the old position is recorded as one node in the flowline and the entire process repeats. This continues until one of the current targets (either a start or goal points depending on which direction we are travelling) is reached.

For each seed point a flowline is traced both forwards and backwards (up and down the gradient) until it arrives at one of the start or end points in the scene respectively (or until it halts as described below). As these start/end points are minima/maxima in the harmonic field the flowlines will always progress steadily towards them. Once a forwards and backwards progressing flowline have both been traced for a given seed point (each of which reaches a start or end point respectively) they are joined into a single flowline which has fully traced the scene and which represents a path within that scene from a start to an end point through the given seed point.

This new methodology is much simpler, and more importantly much quicker. With this approach the only special cases which need to be accounted for are when a move may place the new current point either outside of the defined area of the grid, or into an

obstacle. In both of these cases the flowline drawing is halted at this point and, though the partial flowline produced is kept, it is given a far lower preference for later use than the other complete ones.

This occasional halting means that a flowline is not guaranteed to traverse all the way from the start point to the goal. This is not caused by local minima as, as discussed in Ni et al. (2004), there are no local extrema in the harmonic field. However, there are saddle points in the scene, caused either by obstacles or by points equidistant between several start points. These saddle points are what cause the halting flowlines. This is not a major problem as these failed flowlines are relatively rare and the computational cost of producing them was found to be far less than that of following the more accurate method from Dong et al. (2005). Additionally by finding flowlines and thus discounting using paths along such saddle points, it allows us to prevent the possibility of agents arriving at them and becoming trapped.

The runtime of the flowline calculation is directly related to the length of the flowlines, as it consists of a series of small operations (placing the next node in the flowline based on the current gradient) which only terminate when the start/goal is reached. As a result, it is difficult to give an accurate runtime for this algorithm as it is tied to this length, and the length of flowlines is so variable based on their seed point and the given scene, however it is possible to limit it within some bounds. In the worst case the calculation of the flowline will take $O(x^2)$ where x is the width of the scene. This would occur with a single start point in the top left corner and a single goal point in the bottom right and between them a maze of infinitesimally small obstacles which result in the shortest path between them zig zagging from left to right over and over. Clearly this runtime is quite bad, however this case cannot occur in our system as all obstacles have width. As a result, what we can say is that for any given scene the worst case runtime of the flowline calculation is directly related to the length of the longest path between any start and goal in the scene.

One of the challenges in generating the flowlines, is that it is preferable to have a good spread covering as much of space and encompassing as many of the different routes through it as possible. This is controlled through the choice of the seed points, these are produced in two waves. The first wave is positioned evenly in a circle around each start point, at a radius of r with n flowlines created for each point. Although this provides a good sampling of the directions each agent could travel in from the start, it may fail to provide flowlines which pass through certain areas of the space. This is because some routes from the start points to the goal points may be a lot less direct

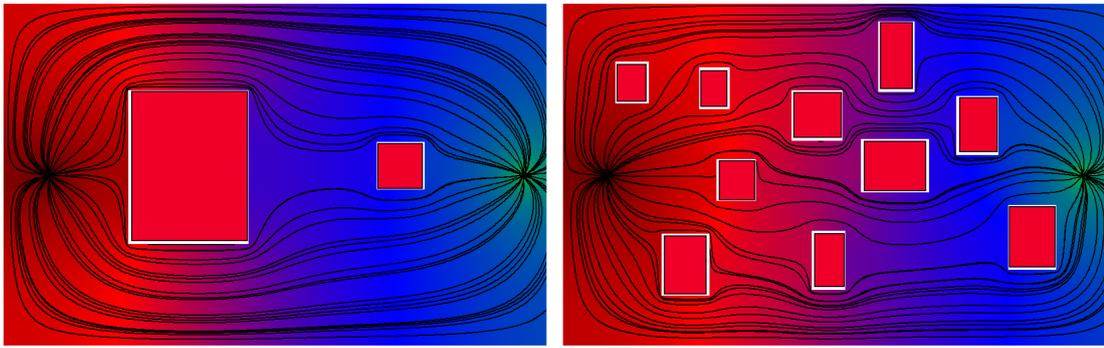


Figure 3.3: The black lines in these two images are flowlines calculated over the Harmonic field gradient using our method with varying configurations of obstacles. In the centre of the right image a flowline can be seen colliding with an obstacle. This is the halting problem described in Section 3.1.4 and it occurs when the seed point for that flowline falls into a saddle in the gradient of the harmonic field.

and, because of the way the harmonic field operates, they may easily fall into a section smaller than the sections which the above generation method samples.

The solution to this is the second wave of flowlines. These are generated from a series of seed points placed in a simple $x \times x$ grid throughout the space. Here x is set to $4\sqrt{n}$, where n is the number of obstacles in the scene. This ensures that for every additional obstacle in the scene there are additional seed points. This is done to guarantee that even for complex scenes there is still a good coverage of the scene. It is also done with a lower bound on the value for x , to ensure that even in empty scenes there is still a good coverage. This grid based seeding ensures that flowlines are generated even in spaces between obstacles which lie roughly perpendicular to the general direction of flow of the flowlines. In practice some of the flowlines generated will not form a connection all the way from a start point to a goal point (due to collisions with obstacles, as described earlier in this section). Equally some of the seed points from the grid structure may fall within obstacles, in which case no flowline may be produced from this point. A good spread covering the space is still produced. The entire set of generated flowlines can be seen for a couple of cases in Figure 3.3.

3.2 Reeb Graph

In this section we will first explain some related work which utilises Reeb graphs. Then we will describe the specific process by which we calculate the Reeb Graph for

our system. Finally we will give examples of the Reeb Graph in action to demonstrate its usefulness.

3.2.1 Related Reeb Graph Work

Reeb Graphs were originally introduced as a part of Morse theory by Georges Reeb in Reeb (1946). Given a topological space X and a continuous function $f : X \rightarrow \mathbb{R}$, the Reeb graph is the connectivity of the level sets of the produced \mathbb{R} .

Reeb Graphs were first introduced to the field of Computer Graphics in Shinagawa et al. (1991) as a means of representing both two and three dimensional shapes in a simplified manner and they have continued to be used for this purpose in the field, as seen in Xiao et al. (2003) and Biasotti (2004). In relation to our work Contour trees (which are actually a simpler subset of Reeb Graphs) were used in Berg and Kreveld (1993) and van Kreveld et al. (1997), in two dimensions and three dimensions respectively, as a means of analysing paths between different points in mountainous terrain.

Multiresolution Reeb Graphs were first introduced in Hilaga et al. (2001). Here the Reeb Graph is computed at multiple resolutions, by dividing the scalar function first into two sets, then four, then eight, up to one hundred and twenty eight. The Reeb Graph methods used in our method have most in common with these as we allow the user to change the resolution for the inclusion or exclusion of smaller obstacles as can be seen in Figure 3.4.

3.2.2 Computing the Reeb Graph

In this section we will describe how the Reeb graph is calculated. Initially we have a mesh over the space with an associated harmonic field value at each node, as described in Section 3.1.2. First these nodes are divided up into level sets of specific field values. Each of these sets describes a node in the Reeb graph and their connectivity to one another determines the edges of the Reeb graph.

The first step in defining the Reeb graph is choosing its resolution, denoted by the value R . This determines how many level sets the field will be divided into and thus how large each of them will be. The value of R can be set by the user if they wish, however, we recommend a default value of $R = 6$. This because if R is too low then local features may not be preserved and obstacles are not recognised, as demonstrated in Figure 3.4. This may or may not be desirable depending on whether the user wants

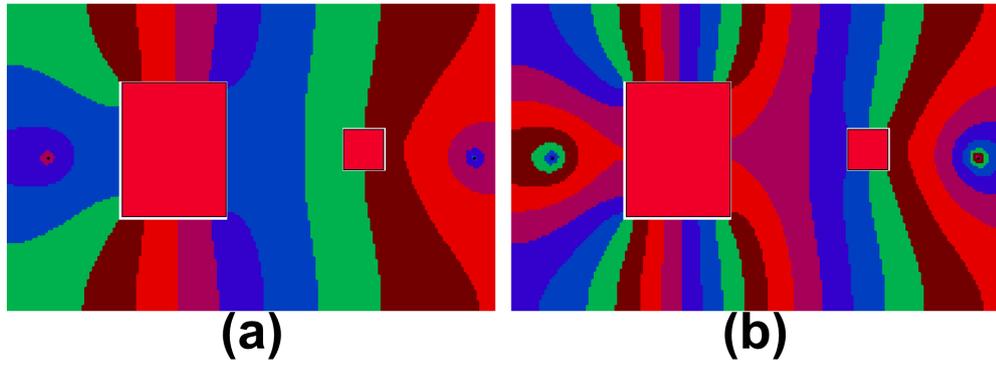


Figure 3.4: The coloured areas here represent the Reeb regions computed using our method for a value of $R = 4$ in image (a) and $R = 5$ in image (b). Here it can be seen that when the regions are connected to form a full graph the right hand obstacle in image (a) will not be registered in the topology of the graph, and it will be fully present in the graph drawn from image (b).

those obstacles to be accounted for by our global planner or by whatever localised planner they choose to use. Generally we found that $R = 6$ is suitable for accounting for almost all obstacles in the cases which we dealt with, however, the option is there if the user wishes to have much smaller obstacles to increase R .

It is also possible to automate the process of choosing R . This can be done if we treat the produced Reeb Graph as a directed graph, with each edge only moving from the lower valued Reeb nodes to the higher valued ones. In this case we can use the following calculation:

$$\gamma = \left(\sum_{i=1}^{i < N} S_i \right) - (G - 1) \quad (3.9)$$

where N is the number of nodes in the graph and S is the number of edges leaving each node greater than 1, that is the splits in the graph, as wherever a node has more than one edge leaving it that represents a change in the detected topology of a scene. Then G is the number of goal points, as each one of these beyond the first will produce another split in the Reeb graph. γ will then be the number of objects which are accounted for in the discovered topology of the scene. Therefore the value of R should be increased until γ is equal to the number of distinct obstacles in the scene (that is, obstacles which do not contact another obstacle or the edge of the scene).

Whatever value is chosen for R there will be 2^R different levels sets in the Reeb Graph (each of which may consist of multiple regions) and each of these levels will span a space of $\frac{1}{2^R}$ of the harmonic field's values.

Now over this mesh the polygons are sorted into a series of sets S_i , where the

polygons in each set have at least one node with a harmonic field value within the region n , where:

$$(i-1)\frac{1}{2^R} \leq n \leq i\frac{1}{2^R}. \quad (3.10)$$

It is worth noting that this division of polygons is not unique. That is, a given polygon may exist both in S_{i-1} and S_i . In Hilaga et al. (2001) they deal with this by subdividing each polygon which spans any edge of a Reeb region. For our system such a solution is not needed as this never causes a problem as long as the Reeb levels are large enough that no polygon spans any three of them. That is, no polygon has nodes which exist both in S_{i-1} and in S_{i+1} . In this case, it could cause issues with the connectivity of the resulting graph. Realistically with the size of mesh which we use this will never happen until R is increased into double figures (and perhaps not even then) and this is never necessary for capturing the detail of any given scene.

Having found the entire series of sets S_i these can now be sorted. First this is done within each i , as each level may in fact consist of several separate regions which are not interconnected. This search process simply involves creating a new region r . A single polygon from S_i is then chosen and added to S_{ir} then each of its nodes which are within the n band for that level S_i are found and added to a list of contained nodes. Each of these contained nodes are checked against the rest of S_i and any polygons which utilise them are considered to be connected and are also added to S_{ir} and their nodes are added to the list. This process is continued until there are no new nodes to be checked against S_i at which point S_{ir_1} is labelled as a full region and, if there are any polygons within S_i which are not within S_{ir_1} , then the process is started again for a new S_{ir_2} .

Having divided S into a series of different regions at each i , the regions must then be connected. This is done starting at S_{0r_1} . Here each of the polygons is checked, but this time for a node which lies within (or on the boundary) of the region of S_1 . If such a node is found then each of the r regions at S_1 is checked and any which have a polygon containing this same node are considered connected to S_{0r_1} and a connection is stored between the two of them in the Reeb Graph. By stepping upwards, through the regions at each i and through the different i levels the entire connectivity of the Reeb Graph is discovered. An example of this can be seen in Figure 3.5. It is worth noting that the graph obtained details the connectivity of the different regions, but once this is obtained it is an abstract graph of topology and the nodes involved do not explicitly contain any information about their position. For the purposes of display in Figure 3.5 the average positions of the polygons involved in each region were used to obtain a

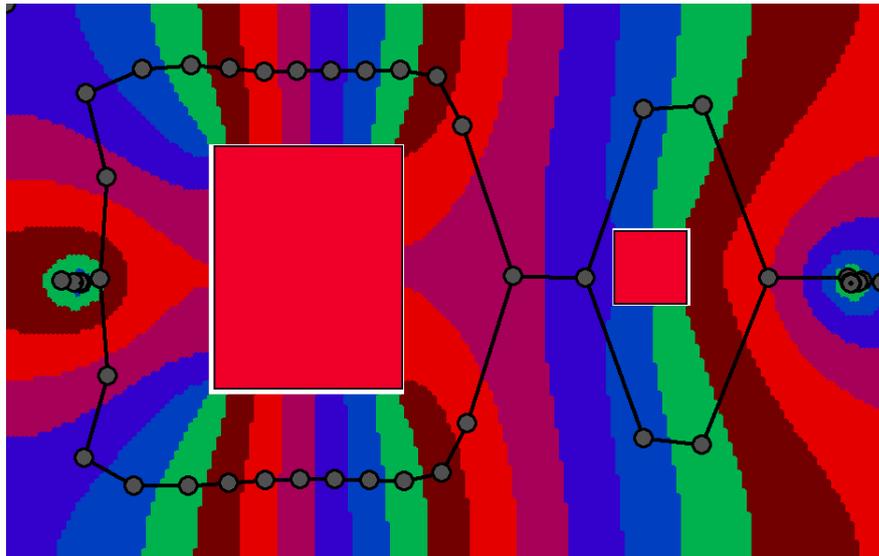


Figure 3.5: An example Reeb graph, with the Reeb regions represented by the coloured areas and the graph shown overlaid above them. Note that although this demonstrates the shape of the graph, these nodes actually have no geometric position, here for the sake of clarity they have been placed in the average position of their corresponding Reeb region.

rough representation of the position of each Reeb Graph node.

In experiments over 100 tests with 2 obstacles the running time of our Reeb Graph calculation was 212ms with standard deviation of 11ms. The amount is largely effected by the number and size of the obstacles and it actually improves as they take up more space (either through becoming larger or more numerous or both). This is because it is computed over the polygons in the open space of the scene and the obstacles obscure open space (that is, they define polygons over which the harmonic field does not need to be calculated). The Reeb graph resolution has very little influence on the speed and what effect it does have is mostly related to storing a larger data structure. The complexity of the algorithm in the worst case is $O(n^2)$ where n is the number of polygons in the space. This is because in the worst case, in finding the connections between reeb regions every polygon will have to be checked against every other one before a single connection is found. In reality the reeb regions will only need to be compared to those above and below so it is much faster. There are faster methods, proposed more recently. The work in Doraiswamy and Natarajan (2009) proposes an algorithm with a complexity of $O(n \log n (\log \log n)^3)$. However, such gains are not necessary as, as also explained in Section 6.2 the Reeb graph calculation is quite a

small part of the overall system's precomputation.

Chapter 4

Maximum Flow

There exist many methods to compute the maximum flow of a graph however, there are some challenges involved in adapting them to this specific problem, as well as in providing the capacities for the graph. In this chapter we will explain these challenges and the solutions we provided to overcome them.

On a given network, with a set of vertices connected by edges and capacities for each of the edges, the maximum flow represents the maximum capacity which the network can take from a given input vertex through to an output vertex. For a given network the maximum flow will always be the same, though there may be many ways of filling up the various edges to achieve that level of flow. This is useful for crowd control because we can abstract the situation of a crowd moving from one place to another to a maximum flow problem. This makes sense as a way of planning the movement of the crowd through a limited space, of determining exactly how they might best plan their paths around the obstacles.

Within our system the maximum flow is calculated over the topology of the given space using the Reeb Graph. The discovered maximum flow values are used to determine both how many agents can move through the system when it is filled to capacity and which edges they will move along in this case. Finding the capacity of a scene is useful because it allows the space to be fully filled if necessary, it also allows defining control for the animator in terms of this capacity. This provides our congestion control which allows a simple value to be used to determine how congested the scene should be, full details on this controller are given in Section 5.5. Equally, knowing the number of agents who will move along each edge in the full capacity case allows the division of the space into a series of routes through which the agents can be directed. The process of assigning these routes is described in Chapter 5.

In the rest of this chapter we will first describe how we find the capacities for the edges of the Reeb Graph using the harmonic field in Section 4.1. Then we will describe the particular methodology of the maximum flow method used on our system, along with some background on maximum flow methods in Section 4.2. Finally in Section 4.3 specific details will be given about how the maximum flow method was altered to better apply to a crowd flow situation.

4.1 Finding the Capacities

In this section we describe how we find the capacities for each of the edges of the Reeb Graph. As previously described, these capacities are necessary in order to be able to compute the maximum flow over the graph. This is another place where the harmonic field described in Chapter 3 are useful in describing the space being used by the agents.

Calculating the capacity of an edge uses a process similar to that described in Section 3.1.4, but rather than computing lines which follow the gradient of the harmonic field, here lines are computed which are perpendicular to this gradient called iso-flowlines. The reason that these are such a good representation of the space available is because, by defining them in terms of the direction which we assume agents will be moving, they tell us how wide the space will be relative to the movement of the agents. The exact process for computing the iso-flowlines is described below in Section 4.1.1.

4.1.1 Computing Iso-flowlines

In this section we describe how we compute the iso-flowlines. First, we explain why iso-flowlines at multiple values are needed for any given Reeb edge, and how those values are computed. Second, we explain the computation process for an individual iso-flowline. Finally, we explain why the methodology for these iso-flowlines differs so greatly from that described in Section 3.1.4.

Each iso-flowline represents the width of the space at a particular point. They are each drawn at each space between Reeb regions for a particular value of the Harmonic field and they follow this value through the Reeb regions. In calculating the capacity of the space between two Reeb regions it is necessary to calculate a number of iso-flowlines. This is because the width of the space between them may vary and, as we are using the iso-flowline to capture this, a number of them are needed to ensure that

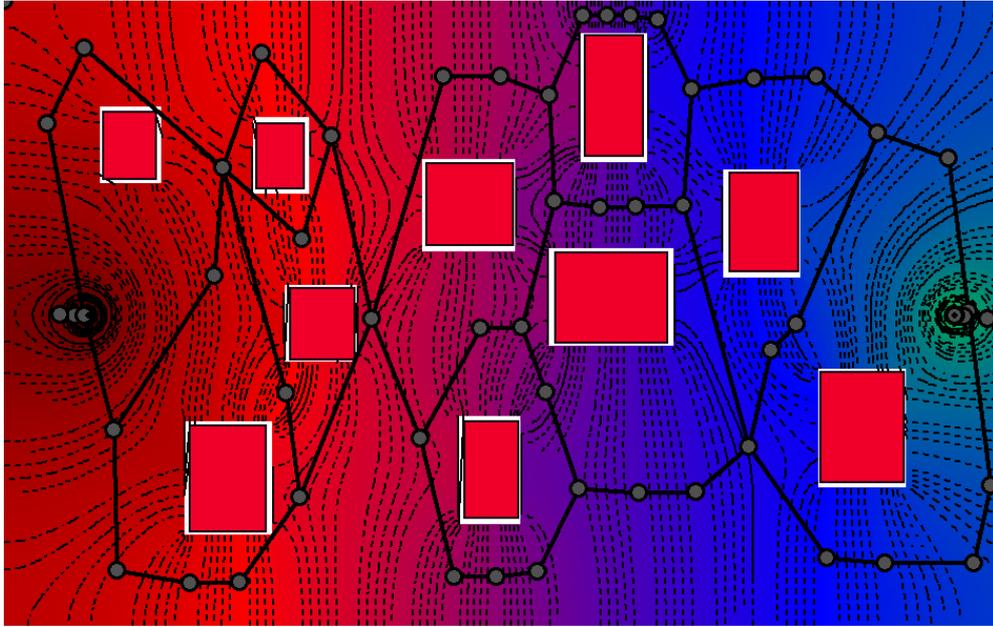


Figure 4.1: An example displaying the iso-flowlines (shown as dotted lines) computed for a given Reeb graph. The spread of iso-flowlines between each pair of Reeb graph nodes can be clearly seen.

the thinnest section is represented.

For a pair of Reeb regions R_i and R_{i-1} which are linked by a Reeb edge, first the average value of each region is found v_i and v_{i-1} (that is, the centre of its range). Then iso-flowlines are computed at the values given below:

$$v_{iso} = v_{i-1} + \frac{x(v_i - v_{i-1})}{n}, \quad (4.1)$$

where x varies from 1 to $n - 1$ and v_i is greater than v_{i-1} . This spreads $n - 1$ iso-flowlines along each Reeb edge, which we found was enough to ensure that an even spread of the area was measured. From these iso-flowlines the length of the shortest is taken to be the capacity of the edge of the Reeb graph between R_i and R_{i-1} . An example set of iso-flowlines computed from this process can be seen in Figure 4.1.

Now we will describe how we compute the iso-flowline for each value of v_{iso} . The first step in doing so is that a complete set of all involved polygons S is collated by combining the sets from the two Reeb nodes (R_i and R_{i-1}). This set is then cut down to a sub-set S_{iso} which contains every polygon from S which has vertices with harmonic field values both above and below v_{iso} (or has a vertex with a value equal to v_{iso}). Any iso-flowline (or collection of iso-flowlines) at the value v_{iso} must involve every polygon in S_{iso} , an example set of such polygons can be seen in Figure 4.3 image (b). Once

S_{iso} is produced a start point is found from within the set and the iso-flowline is drawn in two directions along v_{iso} from that point until it reaches the edge of the allowable space. To create the isoflowline from the starting polygon, first the polygon is marked as having been visited, then the edge in that polygon is found which involves v_{iso} (that is, which has a value above and below it). The new vertex of the iso-flowline is the point at which v_{iso} crosses that edge. The next polygon is then found by finding all of the unvisited polygons involved in that edge. The process can then repeat from that point. The algorithm terminates when no new unvisited polygons have been found, which indicates that either the edge of the allowable space has been reached, or the iso-flowline has looped back around on itself (as can be seen to happen around the start and end points in Figure 4.1). The pseudo-code for this process can be found in Algorithm 1.

Data: S_{iso}, v_{iso}

Result: An iso-flowline through S_{iso}

currentVertex = FindStart (S_{iso});

involvedPolygons = PickStartDirection (FindInvolved (currentVertex));

isoFlowline = [];

isoFlowline \leftarrow currentVertex;

while size (involvedPolygons) > 0 **do**

if size (involvedPolygons) == 1 **then**

This is case (a) in Figure 4.2;

 currentVertex = FindOppositeEdge (involvedPolygons, currentVertex);

else

This is case (b) in Figure 4.2;

 poly = FindNextPoly (involvedPolygons,);

 currentVertex = FindOppositeEdge (poly, currentVertex);

end

 isoFlowline \leftarrow currentVertex;

 SetVisited (involvedPolygons);

 involvedPolygons = FindInvolved (currentVertex);

end

Algorithm 1: The process to compute the iso-flowline in one direction.

In Algorithm 1 the function FindInvolved returns the polygons from S_{iso} which contain the edge along which currentVertex is situated and which have not been visited

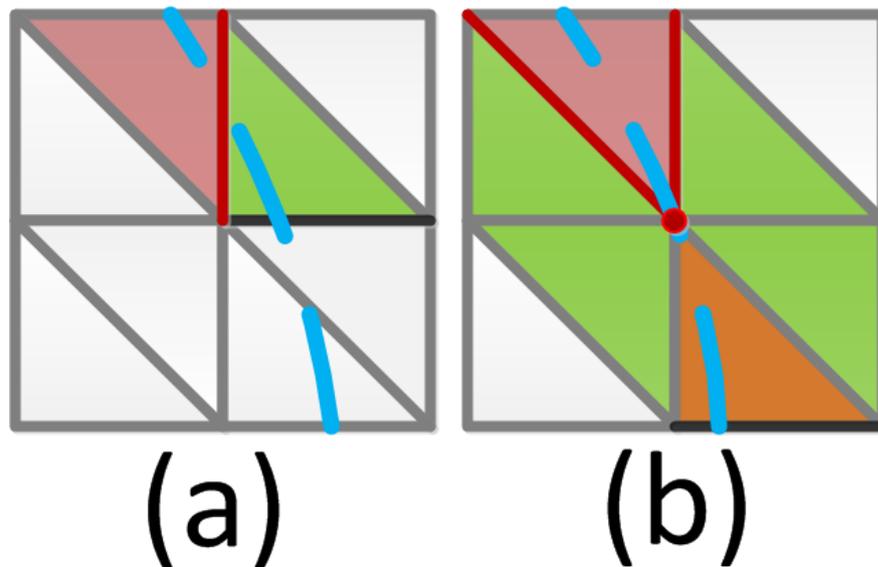


Figure 4.2: The two cases encountered when stepping through the grid drawing iso-flowlines. In both cases the previously visited polygon is coloured in light red, the currently involved edges are coloured in red, the polygons containing the current iso-flowline vertex are coloured green and the next edge found is coloured black. The blue line is the path taken by the actual value of v_{iso} through the space. Image (a) shows the simpler case, where there is only one other involved polygon and only one potential edge to choose from. Image (b) shows the case where the current position of the iso-flowline falls at a one of the polygon's vertices (shown as the red circle). In this case the appropriate polygon (shown in orange) must be chosen from the set of involved polygons such that the next edge can be found.

yet (in both cases these are shown in Figure 4.2 highlighted in green). The function `PickStartDirection` deals with the special case when we have just generated a starting point from S_{iso} using `FindStart`. In this case there will be two edges within the polygon containing the starting point which contain v_{iso} so we must pick one as the direction in which we will first travel (the code shown in Algorithm 1 is used to travel first in one direction and then used again in the other from a given start point, keeping track of the visited polygons across both cases). Given a set of involved polygons as returned by `FindInvolved` there are two special cases to deal with in finding the next vertex for the iso-flowline. These two cases are both shown in Figure 4.2 and are described in more detail in the following two paragraphs.

The simpler and more common case in Figure 4.2, shown in image (a), is dealt with in Algorithm 1 through the if case of the if-else. Here the list of `involvedPolygons`

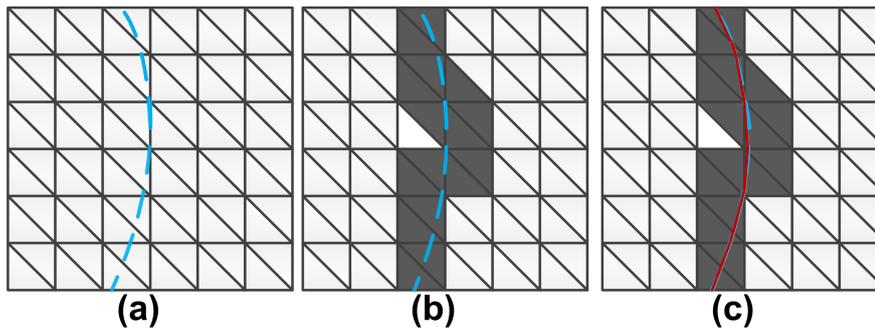


Figure 4.3: These images demonstrate the sequential stages involved in computing the iso-flowline. In (a) there is an iso-flowline following some value v_{iso} through the space but it is not known. In (b) the polygons which involve v_{iso} have been picked out and are shown in dark grey. Then in (c) the actual iso-flowline found by our system is shown in red.

only contains a single polygon and the function `FindOppositeEdge` is called which returns the opposite edge in the polygon to the one containing `currentVertex` which also contains v_{iso} . In Figure 4.2 image (a) this opposite edge is shown in black (with the current edge marked in red). The new `currentVertex` can then be found where v_{iso} crosses that opposite edge and the whole process can begin again to find the next node.

The special case is shown in Figure 4.2 image (b) and dealt with in Algorithm 1 in the else part of the if-else. This occurs when `currentVertex` falls directly on one of the polygons vertices. In this case the set of `involvedPolygons` must first be searched using `FindNextPoly` to find the polygon which contains the next edge (that is the one with values above and below v_{iso} (not counting vertices with values equal to v_{iso})). Once this is found `FindOppositeEdge` can be called to find the next edge the iso-flowline will be crossing.

This process terminates when there are no more `involvedPolygons` which are returned by `FindInvolved`. The list of all polygons S_{iso} can then be checked to see if all of the polygons are marked as visited, if not then the iso-flowline has only been generated in one direction from the start point, and its open end is extended again using Algorithm 1 with the open end used as the start point. Once this is done every polygon in S_{iso} is marked as visited. This allows us to demonstrate the correctness of the approach as we know that if every polygon in S_{iso} is marked as visited then the entire available space at the value v_{iso} will have been covered.

Finally then, the length of each iso-flowline is found as simply the sum of the length of its edges. These can then be compared to give a value for the capacity of the edge

of the Reeb Graph. This value does not currently take the form of any estimation of numbers of agents, instead all the way up until the agents are actually spawned simple scalar values are used. This is to allow the system to be easily scaled such that the number of agents which can fit through a passage can be determined by the size of the agents.

4.2 Computing the Maximum Flow

In this section we will describe the primary method we use to calculate the maximum flow. In our system, at the worst case each new obstacle could double the number of edges in the Reeb Graph. This was potentially problematic as the complexity of most maximum flow algorithms varies based on the number of edges or vertices. After some investigation we found that not only is the magnitude of the problem manageable (a scene with 30 obstacles has a Reeb Graph with only 68 edges and 53 vertices), but also the problems which our system sets up are quite simple for most maximum flow algorithms. This is primarily due to the nature of the problem as, although we do not assume that the edges are directed (that is, that the flow can only pass in one direction along them), in practice the flow found along them nearly always moves from the start to the end points. That is, solutions found nearly always show the flow moving across all edges from the start points to the goal points. This fact means that there are very rarely any cycles of the flow within our system and this significantly speeds up the computation of the Maximum flow.

The precise method used to calculate the Maximum flow was introduced in Boykov and Kolmogorov (2004). This method is used as part of a vision based system to detect edges in images. It is an augmenting path technique, which means essentially that the flow is grown out in a pair of trees from the source nodes and sink nodes until they meet in the centre of the graph. It is suitable for our purposes as, even for examples with a great many obstacles, it runs in a negligible amount of time (for more details see Section 6.2). This method computes the optimal maximum flow, that is, the maximal flow through the scene, however, for our purposes we alter the method such that it may in fact not compute the optimal value, the reasons for this are explained in Section 4.3.

Finally in a basic computation of the Maximum flow we must make a small change for situations where there are multiple start and end points (though as described in Section 5.2.3 even in these cases crossing flows cannot occur). In this case an extra start (and/or goal) is added for the computation of the Maximum flow. These new nodes

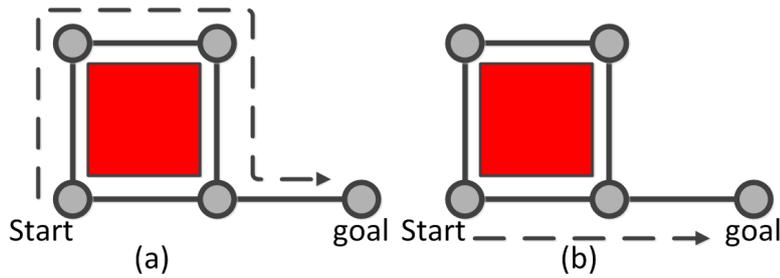


Figure 4.4: Two examples demonstrating the potential problems with maximum flow solutions. Assuming that the capacity of each edge is 1 agent, case (a) and (b) are both valid solutions filling the scene up to a maximum flow of 1. Even though (b) represents a far quicker route for that agent to take.

are connected to the current start points by edges, these are given a capacity greater than the total length of the current space, thus ensuring that they do not influence the calculation of the Maximum flow. This allows the Maximum flow to be found as normal across this slightly extended graph.

4.3 Problems with Pure Maximum Flow

Having computed the Maximum flow, it should then be possible to see how many agents it will take to fill the graph and where they will be moving through the graph in this case. However, there is a problem with it directing where the agents go. Specifically, that there may be more than one way of filling the graph to its Maximum flow and some of these solutions may direct the agents more efficiently than others. This problem can be seen in more detail in Figure 4.4, here it is clear that one way of filling the space is less efficient, though as far as the Maximum flow solution is concerned there is no difference. This problem is exacerbated by our use of the Reeb Graph, as within it any route from a start point to a goal which does not backtrack through the level sets is guaranteed to take the same number of steps through the graph.¹

The solution to this problem is to use the flowlines (whose generation was explained in Section 3.1.4) as a guide. Due to the fact that we know the length of each of the discovered flowlines, we can use these to fill in the capacity of the Reeb graph

¹This is because for a given reeb resolution R there will be 2^R different level sets each representing a different band of harmonic field values. At each one of these level sets there may be many nodes, but as long as a route always steps from the n^{th} level set to the $n + 1^{th}$ level it will always take 2^R steps to get from a start to a goal point. This is problematic for the maximum flow calculations, as it means in terms of the topology of the graph there is no differentiation between a winding route around the outside edge of a scene or a direct one straight through the middle (both will take 2^R steps).

starting with the shortest flowline and working up. For each flowline, we find its route through the Reeb graph and check to find the value of the capacity remaining along it. If this remaining capacity is above zero then that route is filled up to that value, with the capacity of each of the edges being reduced by that amount. This process continues until there are no flowlines remaining, with the flow so far being tracked by adding the amount reduced from the graph each time. Then the Maximum flow is found over what remains using the method described above in Section 4.2 and the amount it discovers is added to the flow so far value to find the total Maximum flow through the graph.

It is worth noting that by carrying out this process with the flowlines it is no longer guaranteed that the actual Maximum flow will be found, due to some edges being pre-filled by the process with the flowlines. Our experiments demonstrated that in practice there was rarely a difference between the original pure maximum flow and our augmented maximum flow. The primary reason for this is that the flowlines tend to only follow the more direct routes through the space, that is, the ones which we would expect to be filled up anyway. This means that by filling up the flow initially by flowlines in this way, what is actually happening is that the shorter routes are being chosen, as shown in Figure 4.4 (b).

Chapter 5

Directing the Agents

In this chapter we will describe how the information condensed in the Reeb graph is converted into a set of paths which are provided for the agents to follow through the space. Thus far the Maximum flow of the graph has been calculated, this describes both how many agents total can fit through the scene and, in this case, how many agents will be moving along each Reeb edge. As yet the actual routes which the agents will take through this graph have not been described. Here the word route is used to describe a sequential list of nodes in the Reeb Graph which a group of agents will pass through, along with a value which describes the number of agents contained within this group. Once a set of such routes has been defined it still remains to convert them to actual paths through the space which can be used to direct and control the movement of individual agents.

In the sections which follow, first, we explain why finding routes through the graph is not a trivial task in Section 5.1. Next, we describe the solution to this problem provided by the RouteTree method in Section 5.2, here first the RouteTree structure is described in Section 5.2.1, then the general methodology is given in Section 5.2.2 and detail about how specific routes are ordered is given in Section 5.2.3. Having created these routes we then describe how they are converted into specific geometric paths for the agents using flowlines in Section 5.3. A potential problem with the use of flowlines and its solution is discussed in Section 5.3.1. Then, in Section 5.4 we explain precisely how the paths produced are followed by the agents. Finally, in Section 5.5 we talk about how our system adapts the information about the routes and their various sizes to provide control over both the congestion and cooperation which occurs within a given scene.

5.1 Route Planning Problems

Currently we have found the available flow over the Reeb graph using the maximum flow calculations, finding a series of routes which fill the graph up to this flow level is trivial. Any series of nodes connected by edges with available flow constitute a valid route, with the size of the group moving along that route being determined by the available flow. Simply moving through the graph along edges with available flow will provide a route, this can be filled with agents following that path, removing that available flow, and another route found. This naive process is guaranteed to fill the space successfully, this is because for every node in the graph the amount of flow moving into and out of it is equal. Essentially the Maximum flow calculations have already done the job of determining how to distribute agents without any becoming stuck. Initially in this work we assumed that this would be enough. However, a serious problem emerged when this naive method was used to fill in this graph, we have termed this problem cross overs.

A simple example of a cross over occurring is demonstrated in Figure 5.1. Here (b) shows provided routes which fill the graph according to the available flow, but when the agents move through the space according to this available flow the two groups will cross over one another's paths. The example in (c) is a set of routes which are equally valid according to the available flow provided by the Maximum flow calculation, but in this case the agents no longer cross over. Given a direction of flow for the agents these crossover situations can be avoided in all cases by having agents who arrived on the left leave on the left and vice versa. The method which we used to do this is described in Section 5.2.

Before describing the solution to this problem, it is worth providing a more detailed definition of why this is considered to be a problem. There is a reasonable argument to be made that in actual crowds such cross overs do occur, therefore it is unrealistic to remove them. There are a number of reasons why it was felt that having such crossovers would be detrimental to the effectiveness of this method. The reason for this, which is that we found that where cross overs occur they cause ugly build up of agents and delay. This runs counter to our desire to provide a dynamic movement for agents through the space and for many scenes where the crowd motion is merely a background, may provide artefacts which undesirably draw attention. As a result we avoid such scenes by limiting ourselves to only those cases where they do not occur, as discussed in Section 5.2.3.

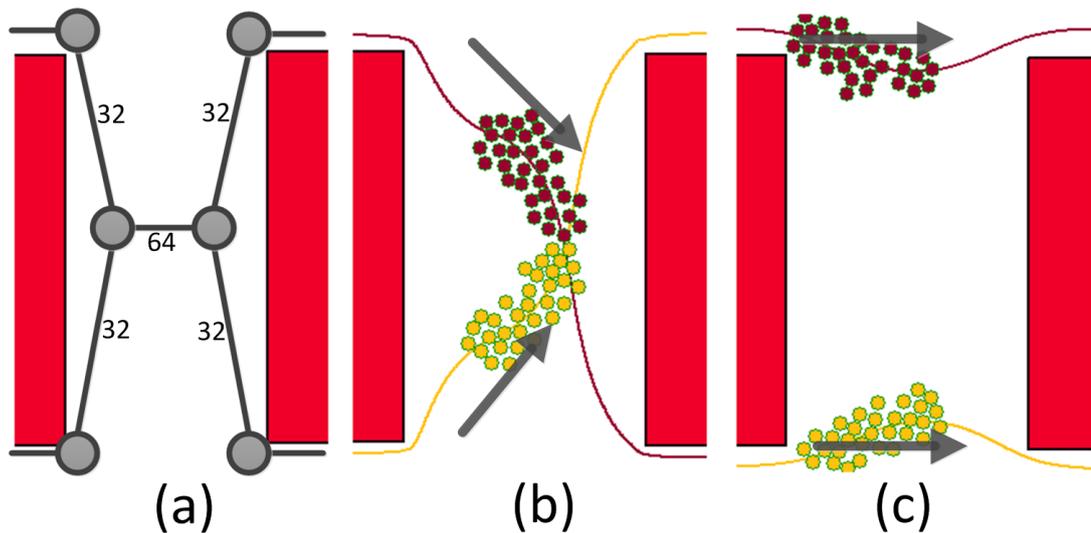


Figure 5.1: This demonstrates the cross over problem. (a) shows the Reeb graph for this scene with the associated flows through each edge. (b) demonstrates one solution to filling in (a) with agents, it results in these two sets of agents crossing over one another's paths. (c) shows the more desirable solution to (a) which involves no cross over.

5.2 Route Planning Methodology

In order to avoid groups of agents crossing over one another's paths some planning is required in converting the Reeb graph and available flow into a set of routes. In this section we describe the method used for this purpose. This is done by stepping through the graph between the level sets of the Reeb graph in a method reminiscent of breadth first search. As the graph is stepped through a record is made of the routes up to that point, this record is kept in data structures we call RouteTrees and these are explained in Section 5.2.1. The process of stepping through the graph is explained in Section 5.2.2. In order to keep the RouteTrees ordered so they do not cause cross over it is important to keep track of where they enter and leave each node relative to one another, the method for doing this is explained last in Section 5.2.3. Finally for further clarification we provide some pseudo-code of the method described in this section in Section 5.2.3.

5.2.1 Using the RouteTrees

The RouteTree is a data structure used to keep track of the different routes which groups of agents take as we step through the graph. It is an ordered tree, with each of

the leaves representing a path through the graph and their ordering defining where they entered their current node relative to one another, from left to right. The RouteTree at a specific node needs to keep track of all of the routes up to that node, as well as how big they are (that is, how many agents can fit along them).

There are two types of RouteTree. The first is a leaf, which simply consists of a route which lists the nodes of the Reeb graph passed through in reaching that point, along with a value denoting how many agents that route can contain. The second is a collection, this consists of a list of other RouteTrees all of which may be either other collections or leaves. The important thing about the collection is that the RouteTrees it contains are ordered, with the order corresponding to where those RouteTrees fall from left to right, as they travel through the Reeb graph. This ordering is explained in more detail in Section 5.2.3. Each collection also has an associated value which shows the total summed size of all of the RouteTrees involved in that collection, that is, the total size of the group of agents who are currently constituted in this collection. An example RouteTree can be seen in Figure 5.2. This is a single RouteTree stored for a particular node of the Reeb graph, showing the different routes used to fill up the graph to that point. Here the routeTree consists of two collections, shown in red, which contain ordered lists of other collections and/or leaves. There are also three leaves shown in blue. The leaves each have a value which indicates how many agents can fit along it and the collections are valued by the sum of the size of their component leaves. The leaves are ordered from left to right. An example graph with the RouteTrees at each stage is shown in Figure 5.3, here the RouteTrees at stage (a) are leaves and the one at stage(c) is a collection.

As the system moves through the Reeb graph, at each stage the RouteTrees will be extended by moving them to a new node and adding the identity of this new node to the route of that RouteTree (or to the routes of each of the leaves involved in a collection). It may also be necessary to combine the RouteTrees into a new collection, if several smaller RouteTrees are moving onto a single larger route, as seen in Figure 5.3 when moving to node d . It is also sometimes necessary to split up the RouteTrees, if a large leaf is moving onto two edges with smaller available flow, as seen in Figure 5.3 when moving to nodes f and g . This process of splitting and combining, along with the details of traversing the Reeb graph are described in Section 5.2.2.

It is worth mentioning the complexity of this process, as it may seem that a large amount of information is needing to be stored. In actuality only the leaves and collections at the current level set of the Reeb graph need to be stored, as they contain all

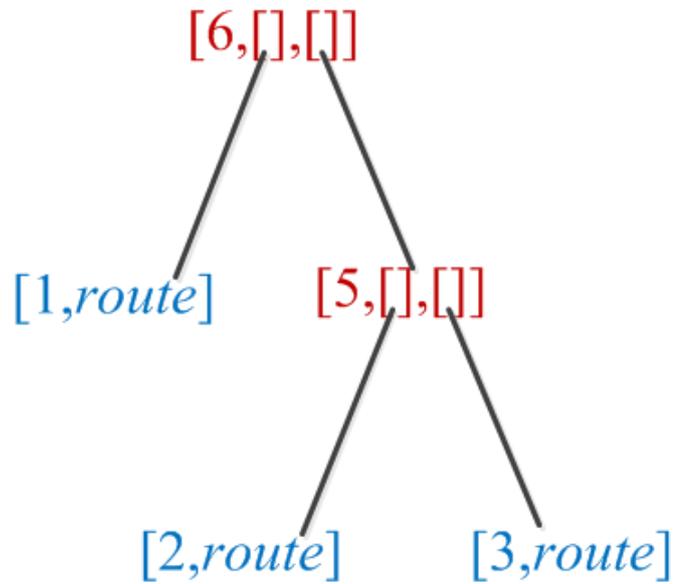


Figure 5.2: An example RouteTree consisting of 3 leaves in blue, with their associated routes. These are ordered into a single RouteTree through two collections shown in red. In both cases the elements are labelled by the size of the group which passes along them to this point.

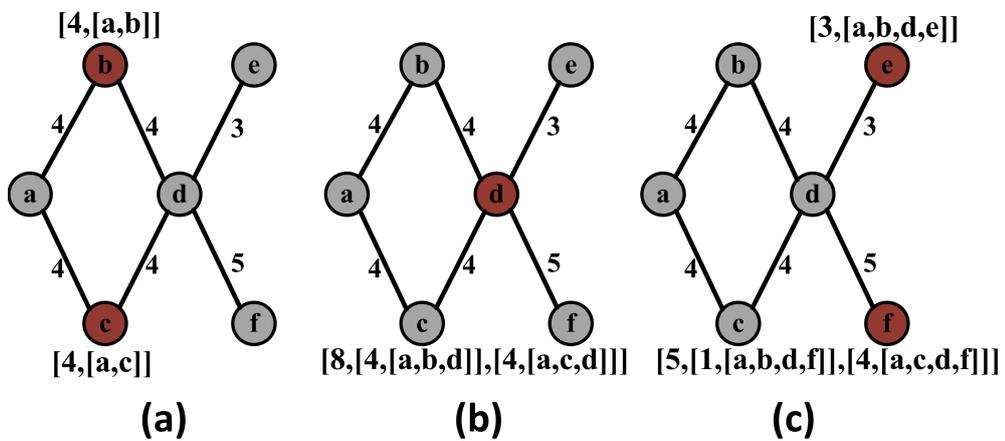


Figure 5.3: The RouteTrees generated at three stages moving through the given graph, each edge of the graph is marked with the flow along it as found by the maximum flow calculation and at each stage the nodes in the current level set are coloured red. The first stage (a) shows the RouteTrees at nodes a and b . The second stage (b) shows these RouteTrees recombining into a collection at node d . Finally the third stage (c) shows how one of the leaves of the collection is split up to account for the variation in flow in moving to nodes e and f .

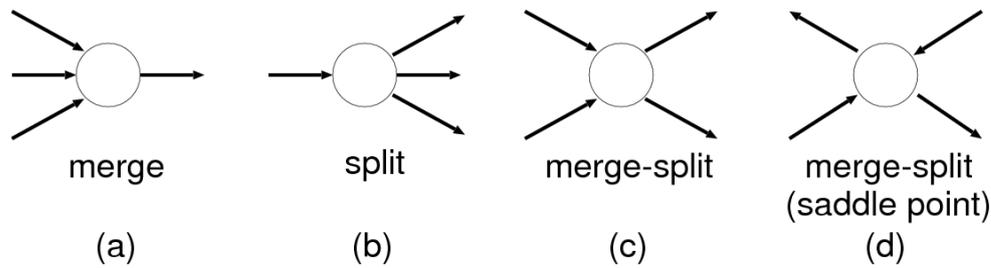


Figure 5.4: This image demonstrates the four kinds of splitting and merging situations which may occur throughout the graph whenever there are more than two edges entering and leaving a Reeb node.

information from the previous steps. Additionally, each obstacle added to a scene only adds at most one split to the graph, this means that in the worst case at the final step there will be a number of RouteTree leaves equal to the number of obstacles (and each of these must have a list of length equal to the resolution of the Reeb graph).

5.2.2 Stepping Through the Graph

The algorithm for stepping through the graph and building the full set of RouteTrees works similarly to breadth first search. Beginning at the start points, at each step it moves down through the Reeb graph to the next level set, that is, those nodes which are associated with the next range of Harmonic field values. At any level set there will be a number of nodes in the Reeb graph and the idea is that for each of these a RouteTree will be built whose components depend upon the connected nodes at the previous level set. It is done in steps in this way because several of the RouteTrees from the previous level set may need to be combined or split to produce one at the current level set (as in Figure 5.3 stage (b) and (c)).

At each new step there are a number of different cases which may arise, these can be seen in Figure 5.4. An example of case (a), the merge case, can be seen in Figure 5.3 when moving from the level set which contains nodes *b* and *c* to that which only contains *d*. Here the incoming edges are ordered from left to right, using the method described below in Section 5.2.3, and a RouteTree collection is built with the incoming RouteTrees placed within it according to their order (from left to right, so here the RouteTree from *b* comes first, then the RouteTree from *c*). Explaining this through the RouteTree in Figure 5.2, we can see that at some previous level set the leaf of size 1 and the collection of size 5 were on different nodes. These were then combined in a merge situation, where two edges merged into this one, and the ordering

was preserved with the size 1 leaf to the left.

The second case in Figure 5.4 (b) is the split case, an example of this can be seen in Figure 5.3 when moving the level set which contains node e to that containing nodes f and g . In this case the outgoing edges are ordered, again from left to right, then the leaves of the RouteTree at the current node are unpacked into these in order. If a leaf is too large for the available flow of the leftmost edge, then it is split into two smaller leaves. This can be seen in Figure 5.3 where the leftmost leaf in the collection at e had to be split between the nodes f and g . The leaves continue to be moved from left of the current RouteTree to the leftmost nodes at the new level set until they have all been accounted for. Explaining this through the RouteTree in Figure 5.2, when we step to the next level set we will preserve the ordering of the agents. That means if stepping to the next level set involves a split scenario, then this RouteTree will be split with the leftmost leaves sent to the left and vice versa.

The third and fourth cases in Figure 5.4, (c) and (d), are initially dealt with by the same process. This is because any time when we are moving to a node which has multiple edges both entering and leaving it, the first thing which must be done is to order all of these entering and leaving edges. This allows us to tell whether this is case (c), in which case all of the entering edges will be next to one another in the ordering and as a result so will the exiting edges. In this case the solution is simply to treat this as a single merge case followed by a single split case using the methods explained above. In the case that the entering and leaving edges alternate in the order then this is a (d) case, these cases are extremely rare but they do occur. For these cases first the system takes note of the involved edges and nodes before waiting for until the next level set. That is, instead of dealing with the nodes at level sets n and $n - 1$, the saddle point cases are dealt with by looking at the nodes at $n - 1$, n and $n + 1$ all at once. Once this next step occurs, the already obtained ordered list of all the incoming and outgoing edges at the n node is stepped through and each incoming edge is unpacked as though this were a split case into any available outgoing edges, if there are any, to its direct left and right. This process is repeated over every incoming edge, then again for all of those edges which still have available leaves in their RouteTrees, until the entire set of RouteTrees have been appropriately allocated. It may seem as though there is potential for this process to go awry, in actual fact because of the Maximum flow calculations we know that the size of the incoming RouteTrees and the outgoing edges at this node will always be equal so there are never any issues.

This process is initialised with a single RouteTree at each start point in the Reeb

graph with their sizes set to the Maximum flow. After the first step of the algorithm they will each have taken on a size appropriate to the available flow from that start point and the entire set of active RouteTree's size will then add up to the Maximum flow. The whole system then proceeds until the final step, at which point there will be a number of RouteTrees at the goal point or goal points. The leaf components of these RouteTrees are split out to form the list of valid routes through the space. These consist of a route of Reeb nodes through which the agent should pass and a value indicating how many agents should pass along that route. The conversion of these routes into actual specific paths for the agents to follow is described in Section 5.3.

To prove the correctness of this route ordering method we can first look at a simple sub example. If we consider the merge-split example in figure 5.4, we know that the flow entering this node will be equal to that leaving it, as it was computed as such by the maximum flow. Thus it will always be possible to order routes entering and exiting the node such that their ordering does not change (that is, those on the left stay on the left). This means that each merge-split case can be treated as only a single node wherein we know the ordering will be kept. Equally the split and merge cases are both simply special cases of the merge-split so the same applies. The saddle point (Figure 5.4 (d)) is a special case. Here the saddle can be seen to be made up of a series of split cases, wherein each incoming node is repeatedly split into the outgoing edges on either side of it. Again by doing so we can guarantee that the left to right ordering of the routes will be kept. This means that the saddle points can equally be treated as a single node within which the ordering is a solved sub-problem. Finally then, as these are all of the cases possible, it can be seen that we can successfully order the entire graph.

5.2.3 Ordering the Edges

In order for the algorithm described in Section 5.2.2 to function it is often necessary to order the Reeb edges connected to a particular region in the Reeb graph. This process is done by traversing along the boundary of the Reeb region producing an ordered list of the adjacent Reeb regions in the order that they are encountered.

Specifically, using Figure 5.5 as an example, here there are four regions connected to the central blue region marked as R_i . The boundary of R_i can be followed by looking for two types of polygon. The first of these are those who have no adjacent connected polygon on the grid, that is, those who are adjacent to an obstacle (as in the bottom of

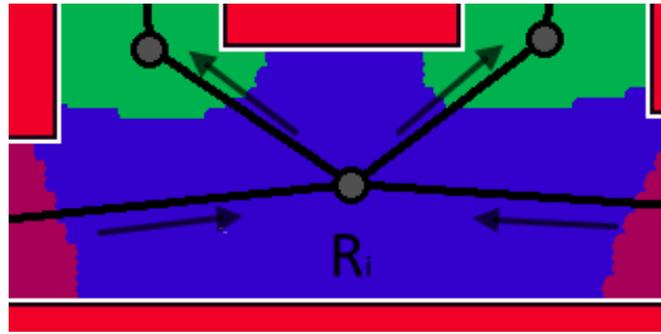


Figure 5.5: An example case where the edges entering and leaving a node need to be ordered. Here the relevant Reeb node is the central one representing the region R_i in blue. The edges are ordered by traversing the edge of this region.

R_i in Figure 5.5. The second are those who exist both in R_i and in one of the adjacent regions, these can be recognised by looking at the Harmonic field values at each of the three nodes they involved. When these are found the Reeb region being passed can be discovered by searching each of the connected region's lists of polygons for the currently traversed one. The discovered region is then added to the ordered list of regions. By traversing the entire edge of R_i in an anti-clockwise direction, checking at each step which type of polygon is being traversed, we produce an ordered list which is used by the method from Section 5.2.2 to avoid cross overs.

5.2.4 Route Planning Pseudo-code

Pseudo-code for the route planning method is provided in Algorithm 2. Here, lines 1 to 5 create the list to contain all the RouteTrees and initialise a tree for each start point using `CreateRoute`, which takes as inputs a value for the capacity of the route (here set to `MaxInt` as we wish the capacity for each route to be limited by the edges along which it travels) and a path (here simply the given start point). The rest of the code is the main loop, and it continues iterating until `Incomplete` returns false on the full list of RouteTrees, that is, there are still RouteTrees which have not arrived at goal points (thus are incomplete).

The first half of each iteration (lines 8-14) deals with the split cases. Here for each RouteTree in the full routetree list the available connections (that is, the reeb edges moving out from the back node of its current path) are found using `FindConnections` and they are then ordered in place from left to right using `OrderConnections` (as described in Section 5.2.3). Then for each connection in the ordered list a new route

is created using `CreateRoute` and the minimum capacity between the `RouteTree` and the new connection. Importantly this line is also where the splitting happens should it be needed. When several `RouteTrees` are combined into one (explained below and included in Algorithm 2 line 22) then their paths are combined into an ordered list of paths, each of which has a number denoting the capacity of that path. In line 12, when the `RouteTree` has a greater capacity than the outgoing node, then the paths it contains are removed from the current `RouteTree` in order and added to the newly created one until their combined capacity meets that of the new connection. Finally in this loop each new `RouteTree` created is added to the hashmap `RouteMap` which maps reeb node IDs to a list of `RouteTrees`, this is used to detect when many `RouteTrees` are arriving at a single node at once and need to be merged.

Finally then the second half of each iteration deals with the merge cases in lines 17 to 24. Here the `RouteMap` is iterated through to find the list of `RouteTrees` arriving at each node in the graph. If there is only 1 such node then it is added to the newly cleared list of all `RouteTrees` ready for the next iteration. If there are many `RouteTrees` arriving at that node then first they are ordered using `OrderConnections`. Next they are combined into one single `RouteTree` using `CombineRoutes`. This function creates a single `RouteTree` whose capacity is equal to the sum of all of the combined `RouteTrees` and whose path is an ordered list of paths from the `RouteTrees` (each with its own associated capacity). This process essentially means that a `RouteTree` can have many `RouteTrees` nested within it (in order) and even further `RouteTrees` nested within them. This combined `RouteTree` is then also added to the list of all `RouteTrees` ready for the next iteration.

5.2.5 Crossing Streams of Agents

The planning described here in Section 5.2 allows us to avoid two groups of agents from crossing one another's paths. However, the effect of this methodology is that with our method it is not possible to produce scenes such as in Figure 5.6 image (a). In the case that our system is given multiple start and goal points in such a manner the route planning will specifically avoid the occurrence of any crossing groups as shown in Figure 5.6 (b). The reason for this is that crossing groups are directly opposed to the type of scene which our system is trying to replicate. Two groups of agents moving through one another's paths will cause congestion and perhaps even queueing, slowing the scene down and dramatically reducing its dynamism. Crossing scenes do occur in reality and there are many methods capable of replicating them, however, we felt there

```

1 AllRoutes = [];
2 foreach StartPoint in StartPointsList do
3   | RouteTree = CreateRoute (MaxInt, StartPoint);
4   | AllRoutes ← RouteTree;
5 end
6 while Incomplete (AllRoutes) do
7   | RouteMap = {};
8   | foreach RouteTree in AllRoutes do
9     | Connections = FindConnections (RouteTree.path);
10    | OrderConnections (Connections);
11    | foreach connection in Connections do
12      | NewRoute = CreateRoute (min (RouteTree.capacity,
13      | connection.capacity), route.path ← connection.node);
14      | RouteMap [connection.targetNode] ← NewRoute;
15    | end
16  | end
17  | AllRoutes = [];
18  | foreach key in RouteMap do
19    | if size (RouteMap [key]) == 1 then
20      | AllRoutes ← RouteMap [key];
21    | else
22      | OrderConnections (RouteMap [key]);
23      | AllRoutes ← CombineRoutes (RouteMap [key])
24    | end
25  | end

```

Algorithm 2: The algorithm described in Section 5.2 for providing a full set of non-crossing routes through the space.

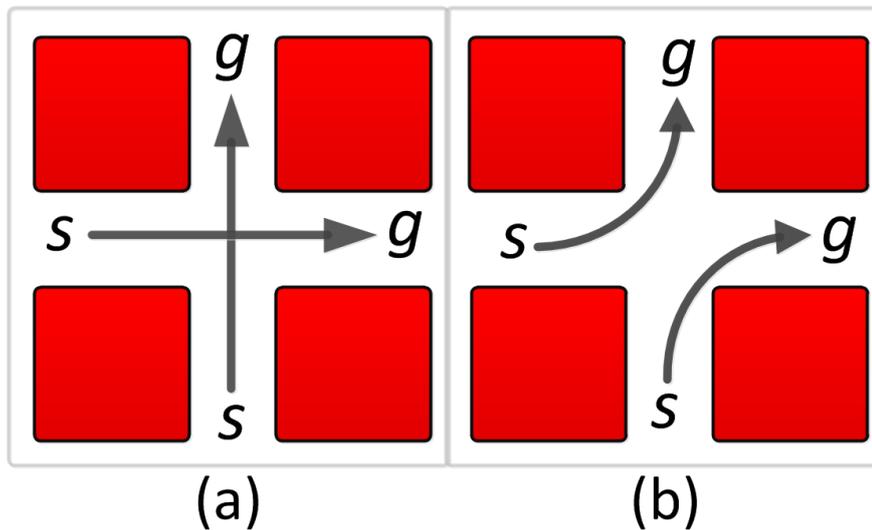


Figure 5.6: An example showing the type of crossing stream situation our system avoids in image (a). Here each s denotes a start point and each g denotes a goal point. Given a scene set up with these points (b) shows how our system will direct the agents.

was no method which allowed the automation of avoiding the crossing of agents, as a result our efforts were focussed on this problem.

It would be possible to create a scene with two or more crossing streams using our system. To do so two or more separate scenes would need to be created and solutions computed, for the example in Figure 5.6 (a), one with the agents moving from bottom to top and one with the agents moving from left to right. The results of these two runs could then be overlaid. To account for the congestion caused by the groups of agents crossing, the flow along each route could be reduced by some percentage depending on how many crossing situations it was likely to be involved in. However, we do not believe that this would be a fruitful application of our system, the crossing groups are bound to obstruct one another making the scene very different from the type which we are aiming to produce. For this reason we have not explored such solutions further.

In conclusion, although it is plausible to create simple examples of crossing groups of agents using our method we do not believe it is advisable to do so as it is not the intention of the work to create such scenes. There are currently many works capable of creating crossing flows. However, producing dynamic scenes which avoid such situations is a much less studied problem.

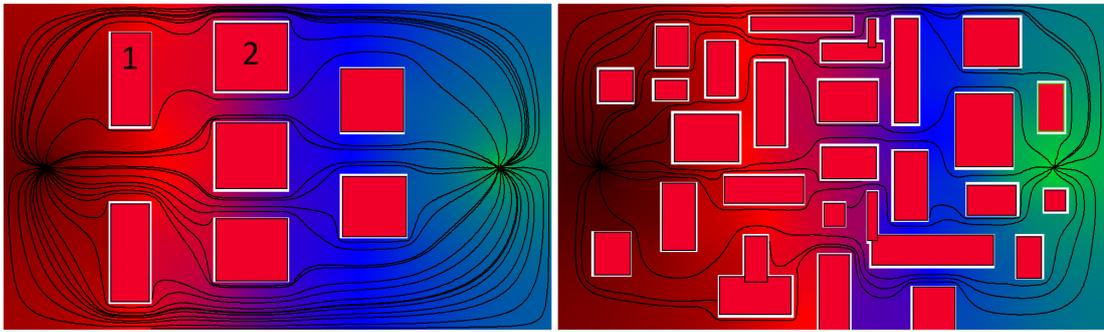


Figure 5.7: A couple of examples showing how the flowlines spread out to fill the whole space. It is worth noting that for ease of viewing only a subset of the generated flowlines are shown here. Also of note is that there are routes through this space for which there is no single flowline following the entirety of the route, for instance any route between the obstacles labelled 1 and 2 in the left image.

5.3 Providing Paths

In Section 5.2 a series of routes were generated to guide the agents through the space, these routes consisted of a series of nodes in the Reeb graph which the agents should pass through. The Reeb nodes do not directly correspond to a particular location in the space, instead they describe a large set of often extremely spread out polygons. As such it is useful to convert this series of nodes into a more exact description of a set of paths through the space which the agents can follow. For this purpose the flowlines, whose computation was described in Section 3.1.4, are used to provide definite geometric paths for the agents to follow. They are ideal for this purpose because they represent smooth and efficient routes through the space which avoid obstacles as well as beginning and ending at the start and goal points. They also spread nicely in open space to use the entirety of the available room. This is demonstrated in Figure 5.7.

Each flowline, as it is being created, records a list of polygons it passes through. These lists are then compared to the polygons contained in each Reeb node region to define a set, for each Reeb node, of the flowlines which pass through it. In most cases for a given route of Reeb nodes, these sets can then be compared to provide a list of flowlines which follow the entire length of that entire route. That list is then used by the local controller to provide paths for the agents to follow when moving along that route. There are rare cases where there is no single flowline which follows the entirety of a route. This cannot be avoided as there will always be saddle points (local peaks) in the gradient of the harmonic field (as in Figure 5.7 on the left between obstacles 1 and 2)

and no flowline will naturally cross these as it would require travelling up the gradient. In this case we require further computation as described below in Section 5.3.1.

5.3.1 Connecting partial paths

It may not be possible to find any flowlines which follow the entirety of a given route through the space. This is normally due to the particular route having a section which moves perpendicular (or nearly so) to the gradient of the harmonic field. For such sections the gradient is normally such that any flowline which progresses into the area or is seeded within it, will end up colliding with an obstacle before escaping the area, an example of this can be seen in Figure 5.7 between the obstacles marked 1 and 2 in the leftmost image. In the case that no single flowline follows the whole of a route. First the route is broken up into sections for which there are representative flowlines. These sections are then stitched together to produce one single flowline which represents the entire route.

In order to stitch together two flowlines, first the two groups of flowlines are found which represent the two sections of the route to be stitched together, as shown in Figure 5.8 (a). Generally each of these flowlines will follow the given route up to some node N or onwards from some node M , in Figure 5.8 (a) N and M are both 3. However, we assume that the portion of the flowline which occurs in N or M will represent a divergence from the route so we instead take each flowline up to node $N - 1$ or from node $M + 1$, this process can be seen in Figure 5.8 (b) with the divergence which is thus removed evident in (a). The ends of these two sets of curtailed flowlines are then compared to find the pair, one from each set, which are closest to one another.

Finally in order to stitch the two flowlines, a new harmonic field is generated with the now end point of the first flowline set to zero (as the start point) and the now start point of the second flowline set to one (as the end point). Finding a new harmonic field in this way is not computationally costly as it is only recomputed for the local area (as defined by the start and end points and any local obstacles). A seed point is then chosen for generating a new connecting flowline, in Figure 5.8 the seed point is chosen from region 3, as the only section between the two sets of flowlines. In some cases there will be a larger break of regions for which there is no representative flowline. In these cases the central region in the break is chosen from which to take the seed point, as this ensures that the newly generated flowline will take the same path around obstacles as the route itself. Given the random nature of the chosen flowline a number may have to

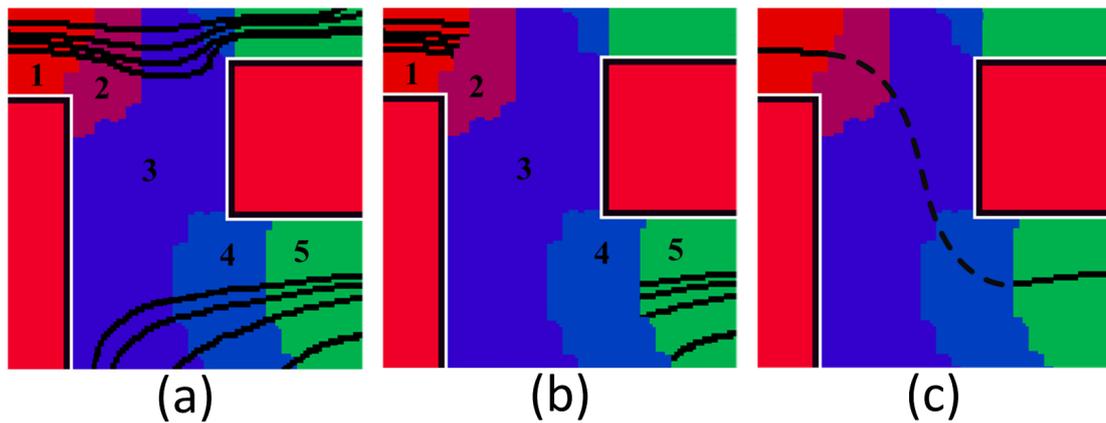


Figure 5.8: This image demonstrates the process of stitching together a flowline for a route which moves between the Reeb regions labelled 1, 2, 3, 4 and 5. First the flowlines representing the beginning and end parts of this region are picked out as shown in (a). Then the relevant sections of these flowlines are picked out as shown in (b). Finally the two closest flowlines are found and a new flowline drawn between the two of them, as shown by the dotted line in (c).

be picked before finding a successful one, that is, one which arrives at the two dangling ends of the current flowlines. Such a successful flowline will connect these two ends up in a smooth way, creating a single flowline which follows this entire section of the route as seen in Figure 5.8 (c). This process is repeated for however many breaks there are in the representation of the route by the flowlines, until there is a single complete flowline which represents the entire route.

5.4 Applying the Paths

For each route through the Reeb graph there is now an associated size and set of paths describing how many agents can travel along this path and where they should move to do so. In order to convert these into actual motion we take a number of smaller steps, all of which are explained in this section. First we explain how we choose the size of the agents and how this effects the overall resulting scene. Next we explain how we exploit the variety of paths representing each route to ensure efficient movement. Finally we explain the controller used to handle the individual agents movement along the paths and collision avoidance as they do so.

Up until this point the actual scale of the scenes being created has only been talked about in very abstract terms, in order to determine the scale of a scene different sizes

of agents are used and they are easily applied to our system. Each route through the graph has an associated width value, but that size is not relative to anything concrete. Agents are assumed to be circular for the purposes of simulation, but it can be seen that depending upon the diameter of these circles a very different number of agents could fit along any route of a given size. For a given scene this should be set such the agents are the correct size relative to the obstacles in the scene. For instance for a scene with four obstacles, the computation is the same whether the obstacles are four trash bins or four buildings and the size of the scene is determined by the size of the agents (for examples see Figure 6.1 for a smaller scale scene and Figure 6.6 for a larger scale scene). The significance of this is that it means the pre-computation time of our system will be the same for a scene involving 100 agents or one with 100,000. For more details on how this value is varied see Section 6.1, which shows some examples of a single scene but filled with agents with different diameters.

Having found the diameter of the agents, this tells us how many agents can move along each route by dividing the size of the route by the diameter and rounding down. In the case that there is a large remainder to this division, that is > 0.5 , this is carried over to the next iteration of the system as an extra agent. At this point it is worth discussing how our system might tackle agents which move at different speeds (or areas of the scene through which agents move at different speeds). This could be dealt with by treating the speed of an agent as being related to its flow, that is agents which are moving more quickly take up less flow (as they spend less time in bottlenecks) and agents which move more slowly take up more flow. This could be formalised if we say that the default speed for agents is s_0 and the speed of an individual agent is s_x , then the flow taken up by that agent will be:

$$F = \frac{s_0}{s_x} \quad (5.1)$$

where F is the flow per agent. The division of each route into a set of agents can then be done based on those agent's speeds and the resulting obstruction which they are going to provide on that route. Equivalently spaces within the scene which will slow the movement of agents can be represented by reducing the available flow in the same manner (by the percentage by which they will slow the agents).

Having performed the calculations to discover how many agents can move along each route we will have some number of agents n to be divided up among a set of paths of size p , the simplest thing to do would be to divide the agents up into groups of roughly $\frac{n}{p}$ and send this number along each path, this is roughly what happens with

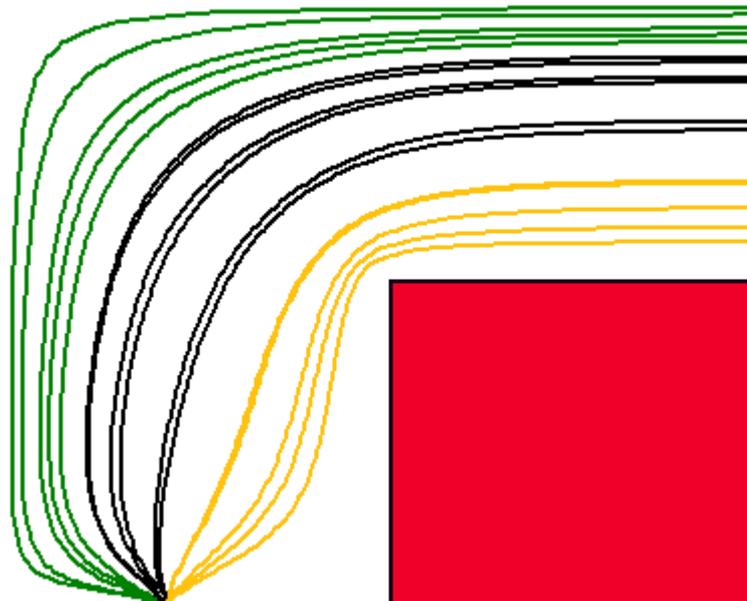


Figure 5.9: This image shows a subsection of a set of flowlines, they have been roughly coloured so that it can be seen that in order to progress around this corner, those in yellow take the most efficient paths and those in green take the least efficient.

one addition. It was observed that for any set of flowlines following a route there will be some which are far less efficient than others, to the point of being much slower. An example of this can be seen in Figure 5.9 where all of the green flowlines are a lot less efficient and some even involve the agents moving backwards to begin with. It is also desirable that as they move through open space the agents remain spread out somewhat. This problem is solved by ordering the list of available paths for each route and only using the shortest half, the n agents being spread as evenly as possible between these. We chose to limit the used routes to the shortest half as this was found to remove the flowlines which took the most obviously inefficient paths while still including enough flowlines to give a good spread of agents through the space. It may also be desirable to discard some of the shorter paths if they are coming too close to the corners of objects. Since we have an ordered list of flowlines it is possible to extend this method to also discard the shortest of these, thus avoiding this issue. However, we did not find it to be a problem in our simulations.

Each agent in the system now has a flowline which it must follow to find its way from the start to the goal. For the purposes of the system it was preferable for there to be as little local control as possible. This is because the entire focus of the work is on providing global controls for such crowds, not on local avoidance and, while there are many local controllers available which could take our paths as input, they nearly all

do at least some form of local planning. As a result there is a danger that, particularly in areas of high interaction between agents, it would be difficult to ascertain which motions were as a result of the direction provided by our method and which were as a result of the local controller. The simplest solution is to have the agents directly moving along the flowlines which they have been provided, with no consideration for one another or obstacles, however, this produces very ugly motion with agents happily intersecting with one another. As a result was decided to use a flocking controller based on the work in Reynolds (1987a) to provide local collision avoidance. The reasoning behind this decision was that we wanted to use as simple as possible a local controller so that the contributions of our own method could be more clearly seen. Finally, as each path originates at the same start point, each agent is added into the scene at some point n along their particular path to ensure that they don't all originate on top of one another. Although there is still some occasional overlap between agents, it is taken care of very quickly by the avoidance portion of the flocking. Each agent then progresses along it's given path, heading to each new point along it in turn until reaching the goal.

5.5 Congestion and Cooperation

In this section we describe how the controls over the congestion and cooperation within the crowd, are implemented. These were described in Section 1.2 in equation (1.3) (repeated below for clarity):

$$X_{paths} = f(scene, agents, starts, goals, \theta). \quad (5.2)$$

The two controls which we chose are the congestion which occurs in the scene, represented by $\theta_{congestion}$, and the cooperation within it, represented by $\theta_{cooperation}$. In this section we describe how both of these can be set at runtime by a user of the system to alter the behaviour of the crowd as a whole. It can be seen that both of these features are implicit in the routes and sizes which have been calculated thus far. The sizes of the routes found over the scene in Section 5.2, describe the situation when the scene is at its maximum capacity without becoming congested. Equally the paths produced in Section 5.3 describe a set of ways through the environment and so, by comparing their lengths, we can discover the shortest, and so most selfish and least cooperative, route to take. In the basic state of the system when each route is being filled the agents are at their most cooperative. These features were specifically chosen because they have a clear intuitive meaning to any user.

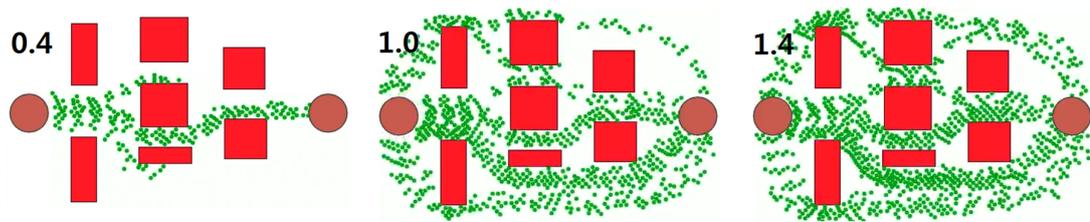


Figure 5.10: Three examples of the congestion value being raised and lowered for a single scene. The congestion value is marked in the top left of each scene.

In its normal run state the congestion value of the system is $\theta_{congestion} = 1$, at a value of 1 the agents being sent into the scene will exactly fill it up to its capacity without causing congestion, as determined by the Maximum flow calculation. The value for $\theta_{congestion}$ can be altered down to a minimum of 0 and up to a maximum of 2 (though the potential maximum is ∞ in reality going above 2 is not practical or useful, as can be seen in Figure 5.10). Mathematically a value of $\theta_{congestion} = 2$ means that the scene is filled to two times its capacity, that is two times more than the maximum flow found for the scene. When the value is altered what actually happens is that the size values for each route are multiplied by $\theta_{congestion}$ for any $\theta_{congestion}$ value above 1 before the agents at that iteration are spawned, creating a larger or smaller number of agents along each route. For values of $\theta_{congestion}$ below 1 the reduction of agents happens from the outer routes first according to a reduction to the total number of agents in the scene. That is, if the total size of all of the routes is 100 and $\theta_{congestion}$ is set to 0.8, then the scene will have 20 less agents moving through it, but these agents will be taken from the longest routes through the scene first. This is because for smaller values of $\theta_{congestion}$ simply multiplying each route results in any routes with a smaller size having very low numbers of agents sent along them, which looks strange especially if those routes happen to be among the shortest and most direct available to the agents in the scene, an example of this can be seen in Figure 5.11. In either case this process happens somewhat in real time, that is the next iteration will cause the scene to be more or less congested as instructed, but the agents currently in the scene will not be removed so it will not instantly alter the current makeup. The results of this process on an actual scene can be seen in Figure 5.10 and further study of the effectiveness of this method can be found in Section 6.4.

The cooperation is represented through the routes found for the agents by our system. Defined by the maximum flow and the choice of routes relating to it, taken together these routes represent the most cooperative situation which the agents could

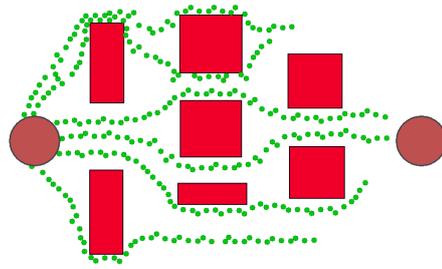


Figure 5.11: An example demonstrating what happens if a low congestion value is used to reduce the agents along each route evenly, rather than reducing the agents from the longest routes first.

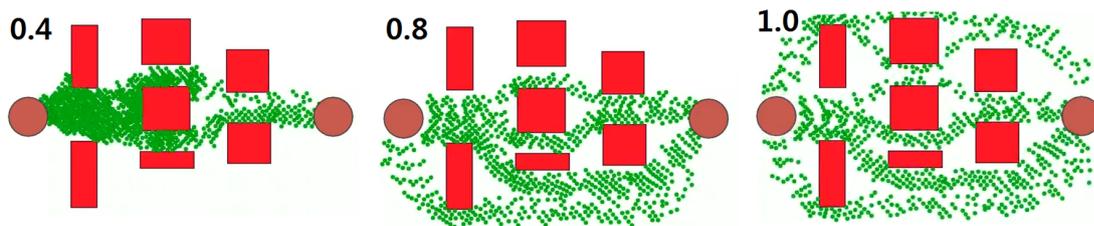


Figure 5.12: Three examples of different cooperation values for a single scene. The cooperation value is marked in the top left of each scene.

utilise to together reach the goal. In the normal state of the system, when all of these routes are being used, the cooperation value is $\theta_{cooperation} = 1$. The range for this value is from 0 to 1, so it can only be reduced from the original value, by doing so making the agents less cooperative and more selfish. When the value is altered what actually happens is that the number of routes used by the agents is reduced by removing longest and the agents who were previously travelling along those routes are redistributed evenly among those remaining, the assumption being that taking the shorter routes is the more selfish option. The routes are removed by first ordering the entire list of routes R according to the length of their shortest associated flowline. The number of agents sent through the space is then reduced, by removing them from the longest routes first, until $\frac{\text{Current agents sent}}{\text{Original number of agents sent}} = \theta_{cooperation}$. Once this value is reached, the removed agents are redistributed among the routes from which no agents were removed. In the case that $\theta_{cooperation} = 0$ (or if it is very small) all of the agents will be removed, they are then all re-added to the single shortest route. This situation, where there is no cooperation between the agents, comes the closest to the type of solution which most other methods would provide for this type of problem. An example of the application of this method can be seen in Figure 5.12 and again further study of its effectiveness can be found in Section 6.4.

Chapter 6

Experimental Results

In order to demonstrate the efficacy of the system laid out this far we performed a number of experiments and explorations. In Section 6.1 we demonstrate the effectiveness of the method in a number of different scenarios. In Section 6.2 we provide some analysis of both the actual computational costs of the system and the theoretical optimal costs, providing a detailed explanation for the discrepancies between the two. In Section 6.3 we provide a comparison to a state of the art method, giving details of both phenomenological and statistical differences. In Section 6.4 further explorations and demonstrations are given of the Cooperation and Congestion controls presented in Section 5.5. Finally, in Section 6.5 we provide some background explaining the difficulty with demonstrating the realism of our method.

6.1 Real World Examples

We have simulated a number of different scenes to demonstrate the running of our system. In this section we will present these examples of our system running on a small town scene and a large town scene, both of which demonstrate the coordination of the agents. Also a large castle attack scene and a monster attack scene, both of which demonstrate the types of dynamic scenes to which our system is best suited. We will also demonstrate the system working across an evacuation scene. Finally to show the scalability of our system we will give an example of it working for an extremely large number of agents. All experiments have done on one core of an Intel core duo E8400 with 4GB of ram. The times given are for providing the full paths for the agents in the crowd to follow, which is all done as a precomputation. Times at runtime are not given as this is not the primary concern of our system. Once the precomputation has

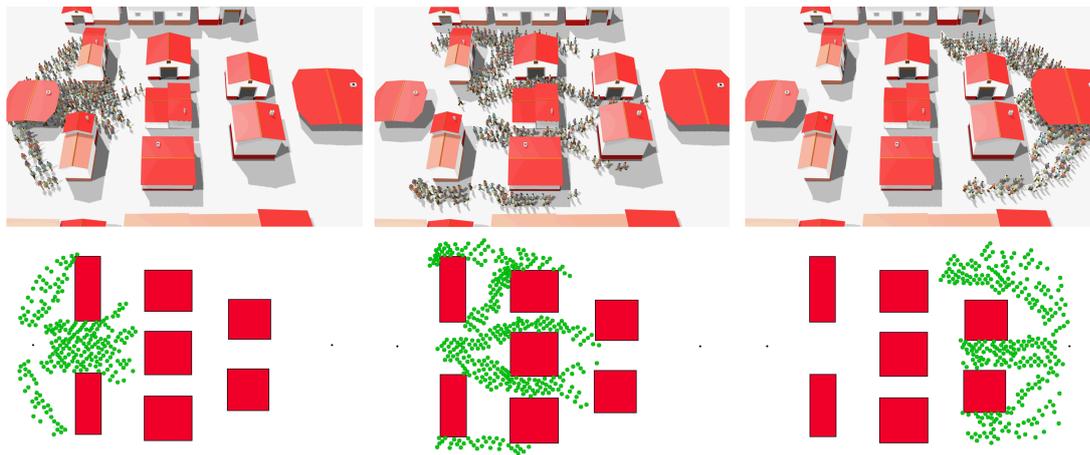


Figure 6.1: An example of a small town scene shown both in our system, below, and animated fully above (animations in this figure and in Figure 6.2 and Figure 6.3 produced in Maya with the kind help of Myung Geol Choi). The agents are moving from a start on the left towards a goal on the right and the images track their progress (again from left to right).

been carried out the only operation performed by our system is to create new agents when necessary and provide them with the appropriate paths through the scene. The entire remainder of the computation is performed by the local controller following those paths. For all examples unless otherwise stated: a Reeb graph resolution R (as described in Section 3.2.2) of 5 was used (giving a Reeb graph with 32 separate layers), and the Congestion and Cooperation values were both set to 1.

The first example shows our system working on a small town scene with a few small buildings through which the agents must navigate, it can be seen both within our system and fully simulated in Figure 6.1. Here there is one start point on the left and one goal on the right, with 7 obstacles between. 324 agents are generated each of whom have a radius of r , from Figure 6.1 it can be seen that the crowd repeatedly splits up to navigate each obstacle, making for a slower route to the goal for each individual agent, but a much quicker one for the crowd as a whole. The pre-computation time for this example is 4.87 Seconds.

The second example shows a larger scale city scene with again a single start and goal point and 26 unique obstacles (where a unique obstacle is one which is unconnected to any other obstacle in the space). Here, to reflect the larger scale of the scene, the agents have a radius of $\frac{r}{2}$, allowing for 1430 to fit within the scene. This example can be seen in Figure 6.2, again the agents can be seen seamlessly splitting up to

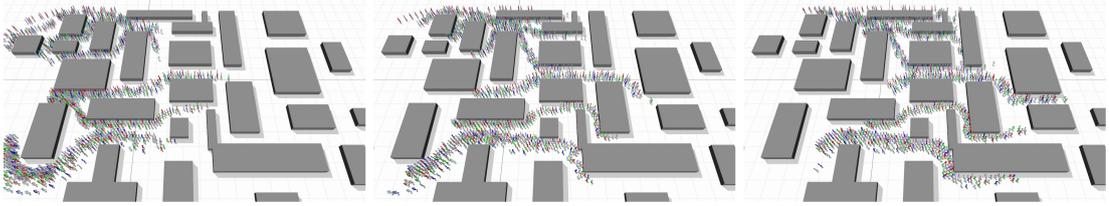


Figure 6.2: An example of a large town scene with the agents progressing through the city from left to right

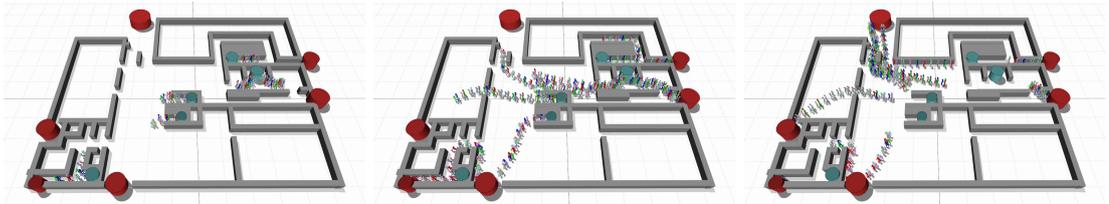


Figure 6.3: An example of an evacuation scene. Here there are a number of start points representing stairs arriving on the ground floor of the building (represented by the blue cylinders) and a number of goal points which are exits from the building (represented by the red cylinders)

navigate obstacles and cooperating in their course through the city. In this example the pre-computation time was 6.69 Seconds (it may seem surprising that this example with more obstacles was actually faster than the previous and slower example, we will provide a full analysis of why this is the case in Section 6.2).

In the third example we demonstrate the effectiveness of our system in cases where there are a greater number of start and goal positions, in this case 5 and 6 respectively. This example, seen in Figure 6.3, is an evacuation scenario. Each of the start points, seen as the blue cylinders, representing the stairs arriving on the ground floor of a tower block and each of the end points, seen as the red cylinders, representing an exit from the building. Here there are 701 agents each again with a radius of $\frac{r}{2}$ and the pre-computation time was 6.64 Seconds.

The fourth and fifth examples both demonstrate the dynamism of our system. In the fourth, shown in Figure 6.4, an army of 3348 agents attack a castle from all sides, while avoiding self obstruction and keeping smooth dynamic flow throughout. Here there are 4 start points around the outsides and 1 goal point at the castle. The pre-computation took 3.34 seconds and the agents each have a radius of $\frac{r}{2}$. In the fifth example, shown in Figure 6.5, there are 297 agents weaving in and out of cars along a street to escape a monster. Here there is just one start point and one goal point. The

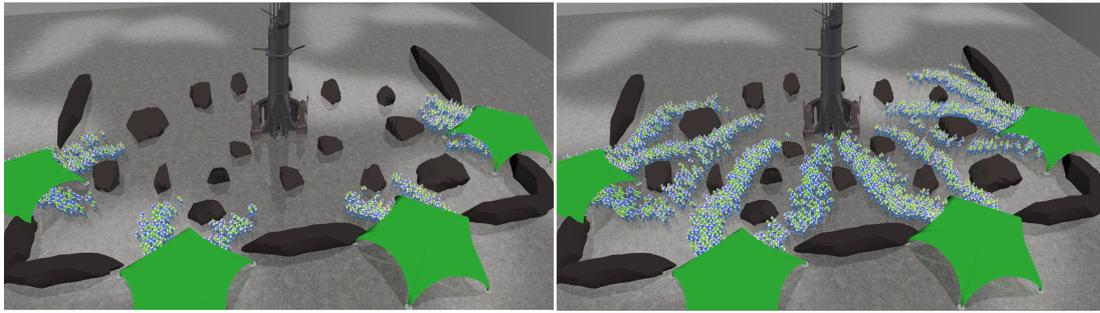


Figure 6.4: A scene with an army rushing to surround a castle, the army is made up of 3348 agents. The four tents signify the four start points and the castle the single goal point.

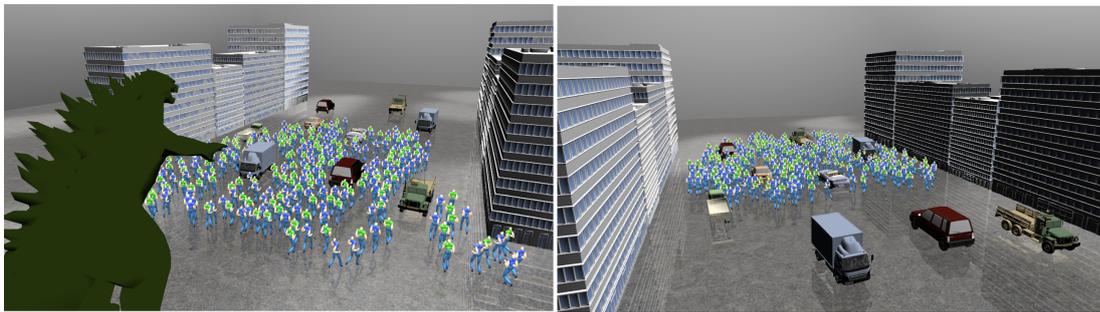


Figure 6.5: A scene with a crowd of pedestrians rushing along a city street away from a monster.

pre-computation took 2.2 seconds and the agents have a radius of r . This example demonstrates the effectiveness of our method on a much smaller scale scene where finer control is needed.

Finally we wanted to demonstrate the effectiveness of the scaling of our method so we created a large scale scene shown in Figure 6.6. Here at its peak there are over 70,000 agents en route between the single start and goal point. The extra scale is achieved firstly by giving each agent a radius of $\frac{r}{6}$. In order to also achieve the level of detailed required to register the topology of the much smaller obstacles in the scene the Reeb graph resolution had to be increased to 7 (128 separate layers). This leads to the slower pre-computation time of 8.86 Seconds. In this example one particularly noticeable feature is the columns or lanes (not to be confused with laning mentioned in Section 6.5) which the agents fall into. This is primarily because although the local controller causes the agents to spread out around the given flowline which they are following, in reality they will still adhere to it to some extent. In extremely high resolution cases such as this the agents are too small to spread out and the shape

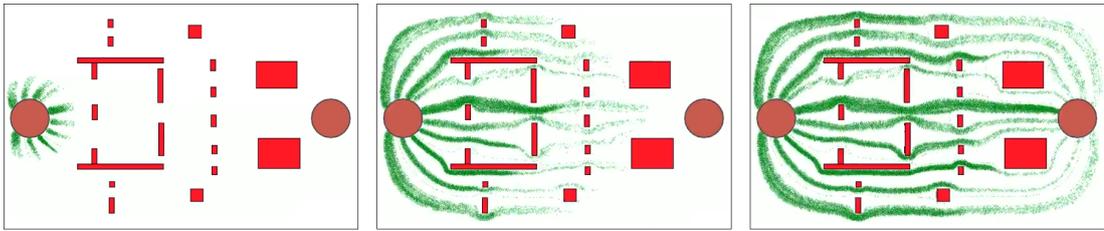


Figure 6.6: A large scale scene with agents with over 70,000 agents, they are progressing from the left to the right of the scene.

of the Harmonic field as it bends around obstacles becomes evident. The result is the lanes which can be seen here. Obviously this is undesirable and could be fixed by creating a much denser spread of flowlines throughout the space, but with the system as it currently is doing so would cause considerable slowdown for reasons explained in Section 6.2.

One thing not mentioned in this section is the runtime computation of the system. As described in Section 5.4 we use flocking for the local control specifically because it is a more basic method, which will not gloss over any of the errors of our system. One of the downsides of this is that it tends to run quite slowly when computing the paths for a large number of agents, particularly for the extremely large scale scene shown in Figure 6.6. The reason for this is that, as the flocking was not a focus of our work it is largely unoptimised and does not cope well with more than 1,000 agents. The runtime speed is solely determined by whatever local crowd controller is used to follow the given paths and there are plenty which are capable of extremely fast speeds for even large numbers of agents.

6.2 Computational Costs

We have thus far provided the pre-computation times for our system, but given little explanation for these or for how they are changed by scenes with a higher level of complexity. In this section we will provide an in depth investigation of these matters. An initial break down of our system can be seen below in Table 6.1, with the computation times and their standard deviations given in milliseconds. These example times were for a simple scene with two randomly placed large obstacles and a single start and goal point and the results were averaged over 100 runs. In the following paragraphs we will examine each of the processes involved in this table, explaining what they involve and why they take the time which they do. For more specific runtime analyses of each

Stage of Pre-computation	mean	std. dev.
Initialising	955ms	21ms
Computing the Harmonic field	765ms	61ms
Creating the flowlines	430ms	26ms
Creating the Reeb graph	212ms	11ms
Finding the Capacities	1131ms	47ms
Calculating the Maximum flow	5ms	7ms
Finding the routes	276ms	111ms
Stitching the routes	29ms	6ms
Total	3805ms	145ms

Table 6.1: The computation times and standard deviations for different stages of our system.

individual component see their individual Sections.

The first steps to look at in Table 6.1 are the initialising step and computing the Harmonic field. Involved in the Initialising step is the allocation for all of the memory for every node, edge and polygon involved in the scene, along with several tables for agents to quickly lookup which polygons they are currently occupying (and ultimately what their associated gradients are). Also included in this are other smaller segments, such as computing the gradients for each polygon in the space along with some simple book keeping operations which re-allocate values throughout the grid. The Harmonic field section computes the harmonic field value for every node in the grid, as described in Section 3.1.2. The most important fact about these two sections are that their computation times only vary in the size of the grid and only linearly with this increase (such an increase might be necessary for an especially large scene, though we never found it so). As a result we do not view these as especially problematic to the scaling up of our system to more complex scenes.

Other stages have an extremely low computation time which is not altered to a significant degree by changing the scene or variables. These stages are creating the Reeb graph, calculating the Maximum flow and Finding the routes. Creating the Reeb Graph collects the polygons of the space into areas with similar harmonic field values, adding connections when they are adjacent, as described in Section 3.2.2. Calculating the Maximum flow finds the maximum number of agents who can travel through this

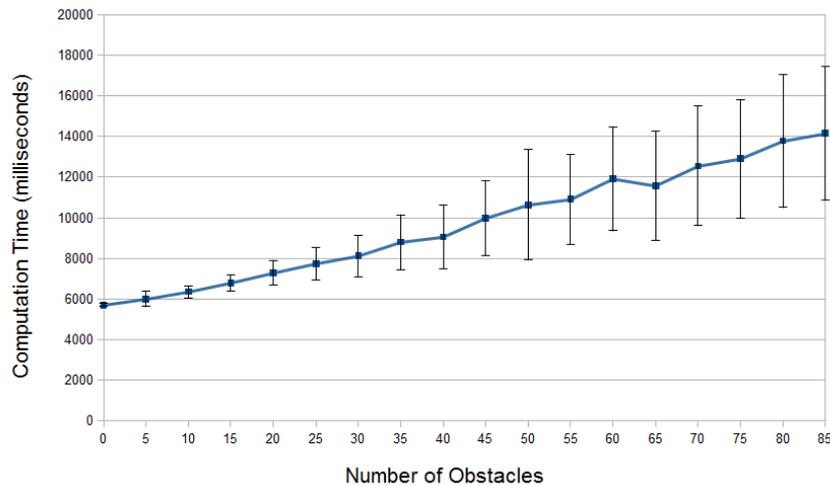


Figure 6.7: This graph demonstrates how the computation time is effected when the number of obstacles in the scene are increased. The standard deviation also shown at each point.

graph, as described in Section 4.2. Finding the routes involves stepping through the Reeb graph finding valid routes for agents to follow along the discovered flow such that they do not cross, as described in Section 5.2. Again these stages are not considered to be problematic as scaling them up to larger or more complex situations causes only a negligible increase in their computation time.

The final stages to mention are: creating the flowlines, finding the capacities and stitching the routes. Creating the flowlines involves drawing flowlines which follow the gradient of the Harmonic field from the start to the goal points, as described in Section 3.1.4. Finding the capacities involves finding iso-flowlines for every edge in the Reeb graph, these run perpendicular to the gradient of the Harmonic field and give an indication of the number of agents who could traverse along that edge, as described in Section 4.1. Stitching the routes involves finding flowlines which match an agent's route through the space. Crucially, when no flowline follows an agent's route the partial ones which do exist will be stitched together, by drawing a new flowline to connect them, this process is described in Section 5.3. These stages all have in common that they compute some kind of flowline through the space. The significant thing about this is that these are the processes which change the most as the number of obstacles is increased and this can be seen in Figure 6.7, discussed below.

In Figure 6.7 it can be seen how the computation time scales with the number

of obstacles. Here the Reeb graph resolution R was again increased to 7 in order to allow it to register smaller obstacles (which were needed to achieve such a high number of obstacles). The obstacles were each of size n by n and were added randomly throughout the space with the only stipulation being that they did not contact any of the already existing obstacles. At each number of obstacles the experiments were run 100 times with a different configuration of obstacles each time. As suggested the vast majority of the increased computation time comes from the increased numbers of iso-flowlines and stitching which increased obstacles require. However, as can be seen from the graph this increase is linear in its effect. The exact source of the increase is that with each new obstacle there will be more edges in the Reeb Graph, which means a larger number of capacities need to be found, additionally with each new obstacle there is an increased chance that there will be some route used for which there exists no single flowline, and so stitching must be carried out. However, it is difficult to predict exactly how each new obstacle will effect the computation, as they increase the variation in the computation time increases, as can be seen in the standard deviation. This is because with obstacles which are alligned with one another (in relation to the start and goal points) the amount of stitching required will be much lower (an effect which is also helped by larger obstacles than are used here). Overall we do not regard this as a problem as the increase is both linear and for the examples which we will be considering (such as those in Section 6.1) it will be towards the bottom end of the variation shown here.

6.3 Comparison

In order to demonstrate the efficacy of the coordination produced by our system we compared it to an implementation of the Continuum crowds method Treuille et al. (2006) as described in this original paper. It was chosen as a suitable candidate partially because it also allows the agents global knowledge of the scene, allowing for a slightly fairer comparison, but partially also because it is highly regarded within the field. The main purpose of this comparison was to satisfy one of the key issues laid out in Section 1.3, that of ensuring the validity of our produced plans. We measured this by comparing the relative success between our system and the Continuum crowds in getting a crowd from the start to the goal as dynamically and efficiently as possible. Here we use the definition of dynamic as laid out in Section 1.1, however for experimental purposes we define our measure of dynamic to be the average movement speed

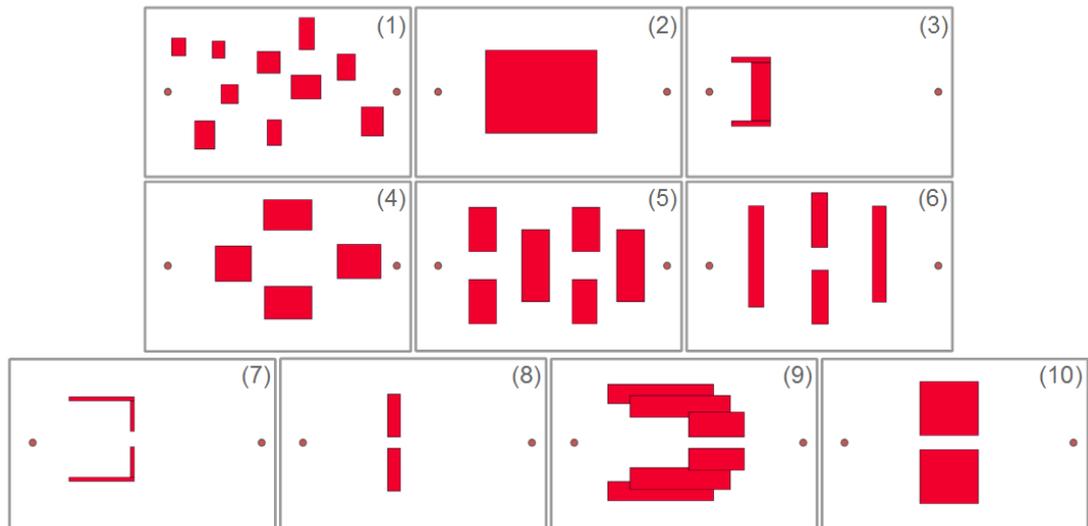


Figure 6.8: The experimental cases on which we compared our system to Continuum Crowds.

of the crowd. That is, a control method should keep each agent moving as close as possible to its maximum motion.

There are other methods against which we considered comparing our method. For example methods such as reciprocal velocity obstacles (van den Berg et al. (2008a)), which are more adept at local avoidance than Continuum Crowds. However, these were not used because our method is primarily global in terms of the control which it provides to the crowd and these are precisely the kinds of problems which reciprocal velocity obstacles is poor at solving. That is, situations in which agents are required to cooperate to progress, as observed in Kapadia et al. (2011b) where they mention that complex interactions and deadlocks cause problems for all of the studied methods (including RVO). Equally there are global methods against which we could have compared, most particularly there are definite similarities between our method and the directable navigation fields presented in Patil et al. (2011). Here they also provide a means to create dynamic motion for the entire crowd. However, the primary difference is that with their method any coordination between agents comes from user input. Therefore such a comparison would result in comparing the automated routes of our system to a set of user input routes provided for their system. Finally, comparison to local agent based methods would be undesirable, primarily because we regard our system not as a replacement but rather as a compliment for such approaches and a hybrid of our approach with these would be an interesting further project.

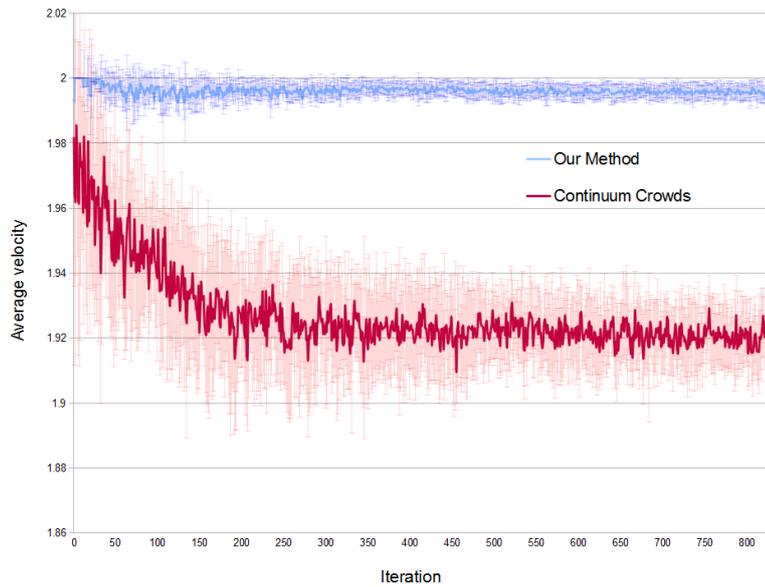


Figure 6.9: Shows the magnitude of the velocity averaged across every agent in every scene shown in Figure 6.8 at each iteration. The standard deviation is plotted in the paler lines for each iteration.

The comparisons were made across a number of example cases shown in Figure 6.8. In each case there is a single start point on the left and a single goal on the right. These cases were chosen to represent a particular set of situations. Specifically, examples 7, 8, 9 and 10 in Figure 6.8 all represent a form of bottleneck which requires forward planning to avoid and account for, a situation in which we feel our method will heavily outperform Continuum Crowds. Examples 4, 5 and 6 all represent cases where the natural flow of the crowd around the obstacles should be close to the maximum flow through the scene, that is they are scenes where we feel our system and Continuum Crowds ought to perform quite closely. Finally examples 1, 2 and 3 are designed to check both systems on a few individual cases, respectively they are: a larger case where a central route is available but where outer routes will need to be utilised for maximum efficiency, a single obstacle where the choice about splitting the crowd needs to be taken early on and finally an example of a simplified scene with a non-convex obstacle.

We compared the average velocity of all of the agents across every experimental case as a measure of the dynamism of the entire crowd motion and the results can be seen in Figure 6.9. Here the maximum velocity for an agent is set to 2, a value which our system manages to maintain across the entire course of the run. Continuum crowds however whilst it starts well, quickly slows, this is because it sends all agents along

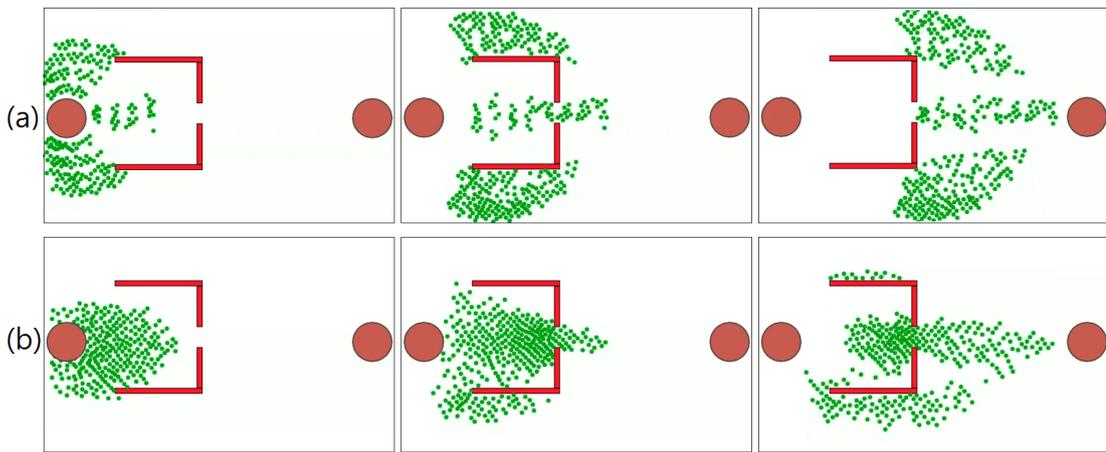


Figure 6.10: Snapshots of our system (a) and Continuum crowds (b) running on the example 7 from Figure 6.8.

the optimal route, even when there is not enough space for them all to fit through that route. Additionally, due to this lack of planning, there is a much greater variation in the speed provided by Continuum Crowds from run to run, as can be seen by the standard deviation. An example of this in action can be seen in Figure 6.10 which shows both systems operating on the top left experimental case. Here it can be clearly seen that our system plans around the bottleneck in the centre where Continuum crowds does not.

It can be seen that in Figure 6.9, though our system keeps the agents far closer to the optimal velocity, the absolute difference in velocities is quite small (only about 0.1). The reason for this is that the Continuum crowds method is good at keeping constant velocity throughout the crowd. If we instead compare the velocity towards the goal, as seen in Figure 6.11, the difference between methods can be more clearly seen. These values represent the progress of agents towards the goal on the right and penalise the creation of situations where agents are forced to backtrack (creating velocities which move away from the goal). Looking at this figure it can be seen that our method provides a much larger improvement over Continuum crowds, as all of the motion is in the direction of the goal, whereas with Continuum crowds much of the motion of the crowd involves agents moving away from the goal as they backtrack from points of congestion. Additionally there is a much greater variation in the control provided by Continuum crowds, particularly towards the beginning of the run and around the 300 to 350 iterations mark. The reason for this latter increase is due to the bottleneck problem discussed previously and it can be seen in much more detail in Figure 6.12. Here we provide the results for only the bottleneck cases. It can be seen that Continuum

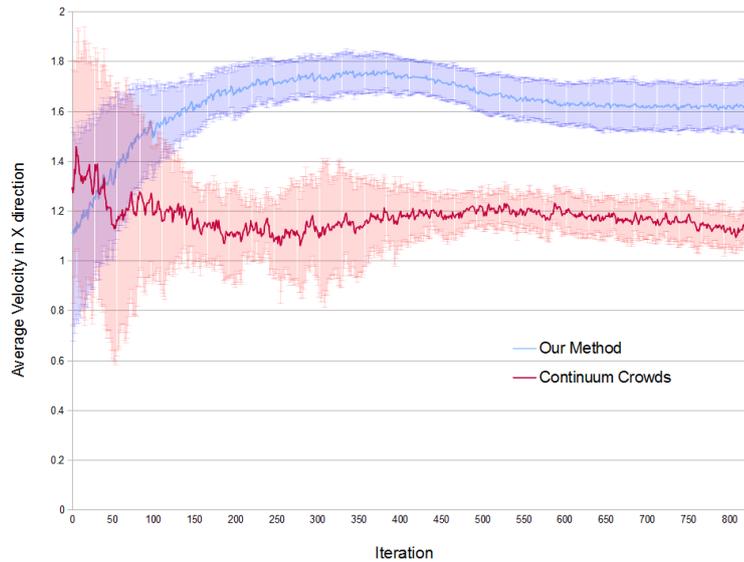


Figure 6.11: Shows the x component of the velocity averaged across every agent in every scene shown in Figure 6.8 at each iteration. In this case agents moving backwards will have a negative value, lowering the average. The standard deviation is plotted in the paler lines for each iteration.

crowds performs much better initially, even outperforming our method for the first 100 iterations. However, after this there is a significant drop off which continues below an average x velocity of 1. This defines the point at which the agents controlled by Continuum Crowds all arrive at the bottleneck and are unable to proceed beyond. Note that, even though these are only the bottleneck cases, there is still a large variation in the motion produced by Continuum Crowds from 250 to 400 or so iterations, this is because some of the bottlenecks are more difficult to navigate around once encountered (case 7 in Figure 6.8 as compared to case 8 for example).

6.4 Cooperation and Congestion

In this section we will demonstrate the effects of the Cooperation and Congestion values on our system. The precise details of the operation and application of these values were described in Section 5.5.

The Cooperation value, C_{oop} , describes how much the agents cooperate globally in their progress to the goal. The value ranges from 0, where there is no cooperation, to 1, where they are fully cooperating and taking advantage of every route to fill the space to its full capacity. A range of values can be seen demonstrated on both the small

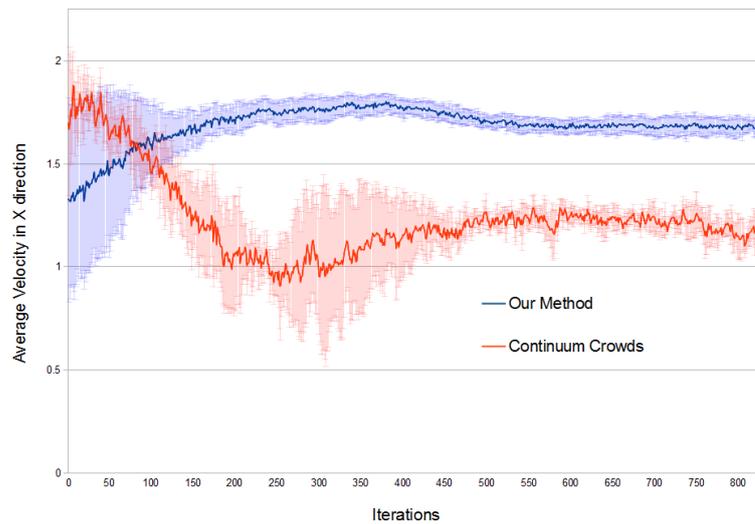


Figure 6.12: Shows the x component of the velocity averaged across every agent in the scenes with bottlenecks shown in Figure 6.8 (scene numbers 7, 8, 9 and 10). As in Figure 6.11 agents moving backwards will have a negative value, lowering the average. The standard deviation is plotted in the paler lines for each iteration.

town scene in Figure 6.13 and on the large town scene in Figure 6.14. Here it is clear how reducing the value of C_{oop} effects the degree to which the agents cooperate with one another. It can be seen how lower values divert agents from the longer routes to shorter ones, which are more immediately beneficial to them, and how this diversion is detrimental to the crowd as a whole, with very large gridlock appearing in the shortest routes.

The Congestion value, C_{ong} , describes the amount of congestion in the scene. In this case it is not an effect on the type of behaviour between the agents but rather on the type of crowd produced. Setting the value of C_{ong} to 1 fills the scene to its capacity, values below 1 remove agents to produce a more sparse and less congested scene, values above 1 fill the scene above capacity such that it becomes congested and gridlock occurs around the bottlenecks or points of lowest capacity. Theoretically there is no upper limit on the values which could be assigned to C_{ong} , but in practice we found that increasing it beyond 2 simply congested the scene well beyond the point of any usefulness, so for this reason this was chosen as an upper limit. A range of values of C_{ong} can be seen demonstrated on both the small town scene in Figure 6.15 and the large town scene in Figure 6.16. Here it can be clearly seen how lower values lead to an emptier scene, while values above 1 lead to extreme crowding and even gridlock in the more extreme cases. There are a few features worth mentioning, firstly values

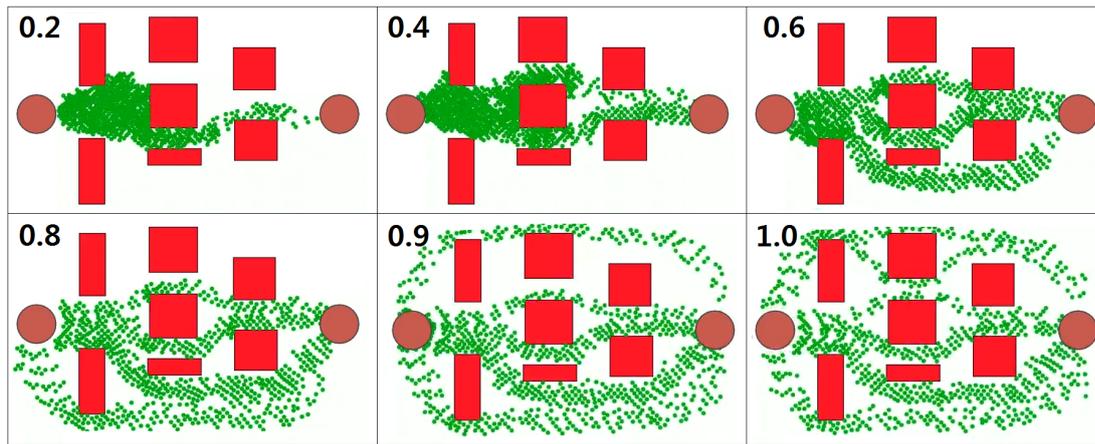


Figure 6.13: Example runs of our system on the small town example, with 6 different values for the cooperation variable.

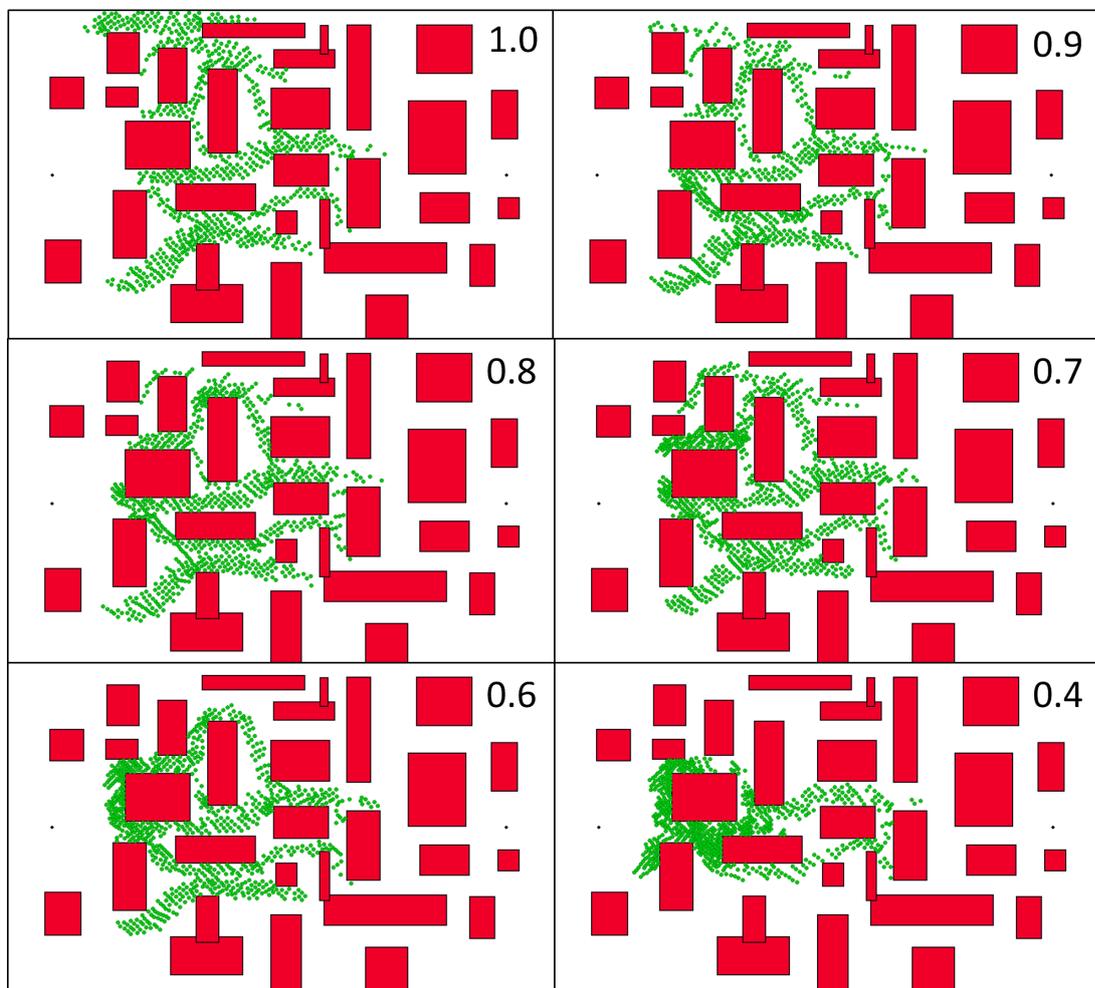


Figure 6.14: Example runs of our system on the large town example, with 6 different values for the cooperation variable.

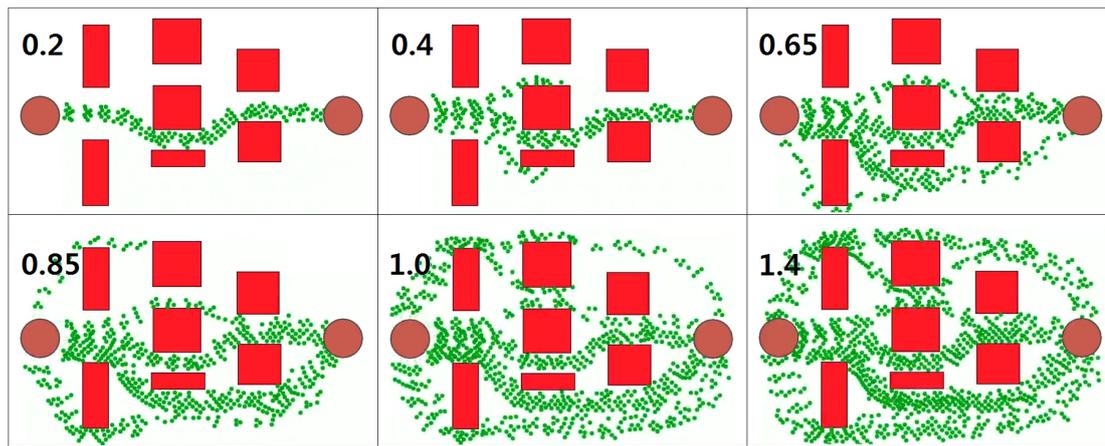


Figure 6.15: Example runs of our system on the small town example, with 6 different values for the congestion variable.

below 1 reduce the crowding by removing agents from the longer routes first, rather than reducing the crowding across all routes at once. This approach was taken because reducing across all the routes evenly quickly led to a very small number of agents taking long routes around when there was space on shorter routes and this produced an artificial seeming effect as shown in Figure 5.11. Equally it is worth noting that even in the extremely congested cases the agents still make good progress through the scene, this is because in such crowded cases the flocking method does not operate well and sometimes allows agents to intersect. Even so the effects of producing a more congested scene are still obvious in these examples.

6.5 Realism of the system

One of the primary concerns of any work in computer animation is that it needs to be realistic. That is, it cannot look obviously wrong to an observer. The standard within crowd animation systems is to demonstrate realism by providing evidence that certain phenomena observed in actual crowds are also reproduced by the system. We will look at the two most popular of these phenomena and explain why they are not valid measures of the realism of our system.

The first real world phenomena is laning. That is, when two opposing groups of agents intersect, as shown in Figure 6.17. In these cases the agents moving in either direction form lanes, with the lanes allowing some continuation of the motion and preservation of the initial group structure while still giving way for the opposing group to also continue forward momentum. This is a very well known example and

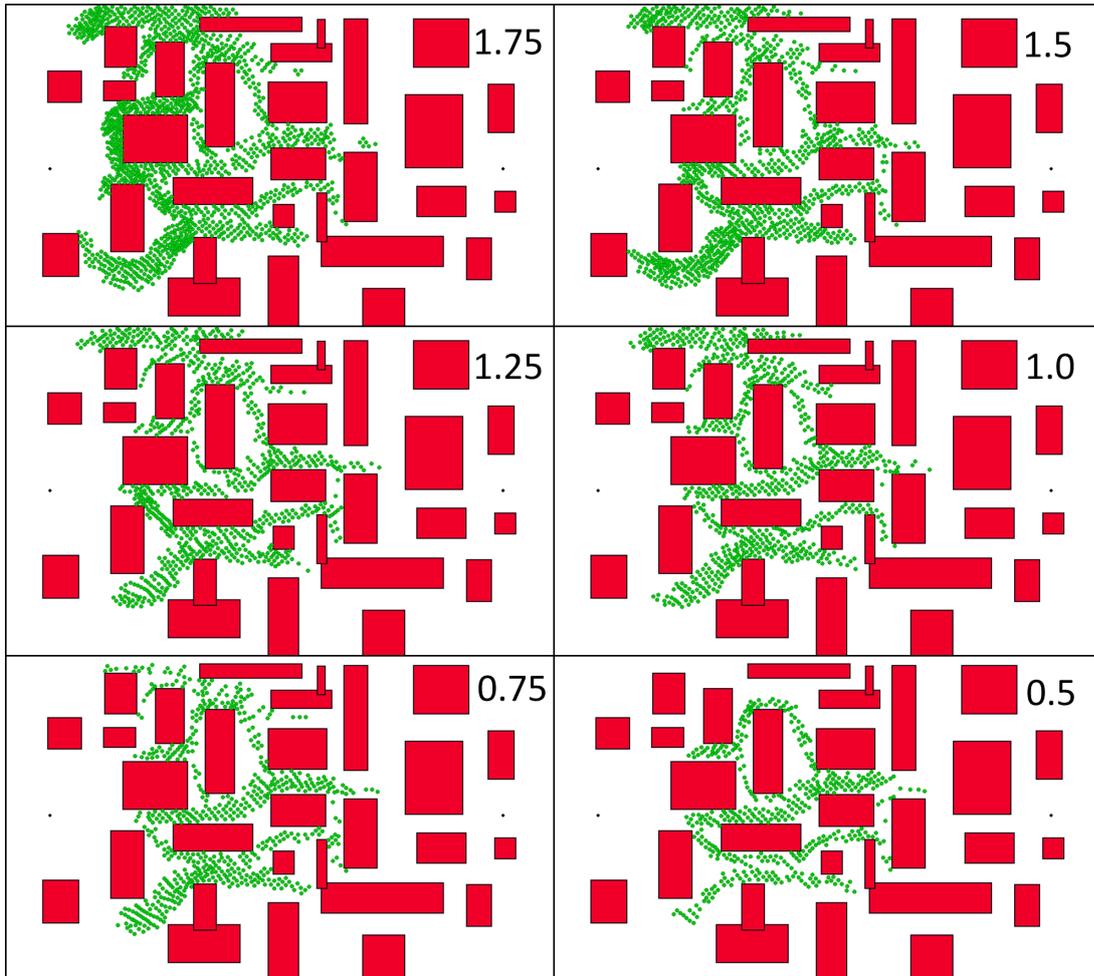


Figure 6.16: Example runs of our system on the large town example, with 6 different values for the congestion variable.

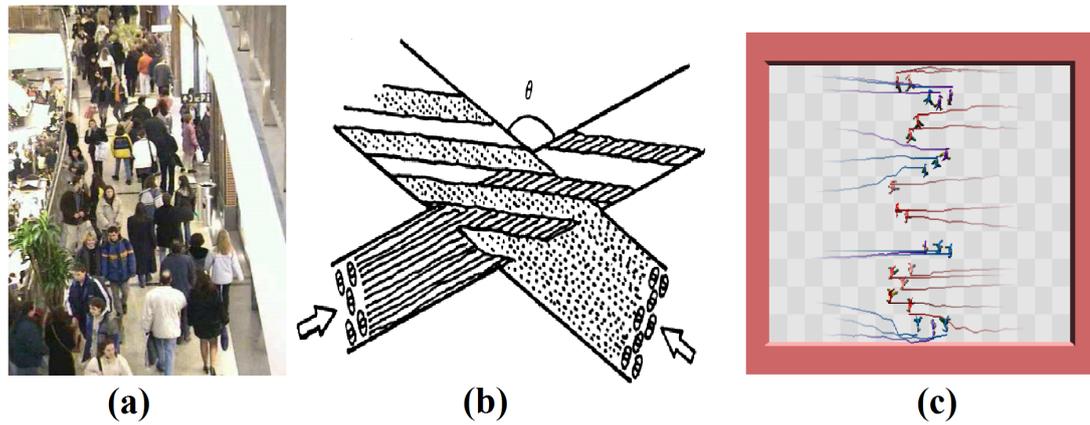


Figure 6.17: Three examples of laning formed in crowds. Image (a) is a real world example, (b) is a diagram demonstrating how this occurs, these are both taken from Helbing et al. (2003). Shown in (c) is a reproduction of laning by the Continuum crowds system, taken from Treuille et al. (2006).

recreation of it is taken as a good indication of the realism of the model. Sadly it is a localised phenomena, that is, its occurrence depends not on the agent's global plan, but on the local avoidance of other agents. It would be possible to produce situations like Figure 6.17 (b) in our system, but where the flowlines being followed by the agents intersect, the formation of lanes would be entirely dependent on the local planner. Currently we use flocking, as described in Section 5.4. As a result of this locality, attempting to create laning within our system would not evaluate the realism of our model, but of whichever local planner it was used in conjunction with.

The second real world phenomena often used to demonstrate realism is vortices and these do deal with a more global interaction. These occur when a number of groups of agents greater than 2 (normally 4) intersect at once, with their paths all crossing some central point. Here the popular understanding seems to be that a vortex should form as all of the agents move past one another, giving the entire crowd a rotational motion, as seen in Figure 6.18. However, it appears that this is not in fact a real world phenomena, rendering its recreation far less meaningful. The most definite reference we were able to find was in Treuille et al. (2006), where Helbing et al. (2003) is cited as evidence that vortices are the realistic result of these kinds of multi-group intersection. Looking at Helbing et al. (2003) it appears that this is a misunderstanding as they specifically state that “no stable pattern exists when three or more pedestrian streams intersect“. The misunderstanding is widespread and it is used to measure the validity of the system by many papers, Treuille et al. (2006), Narain et al. (2009) and Guy et al.

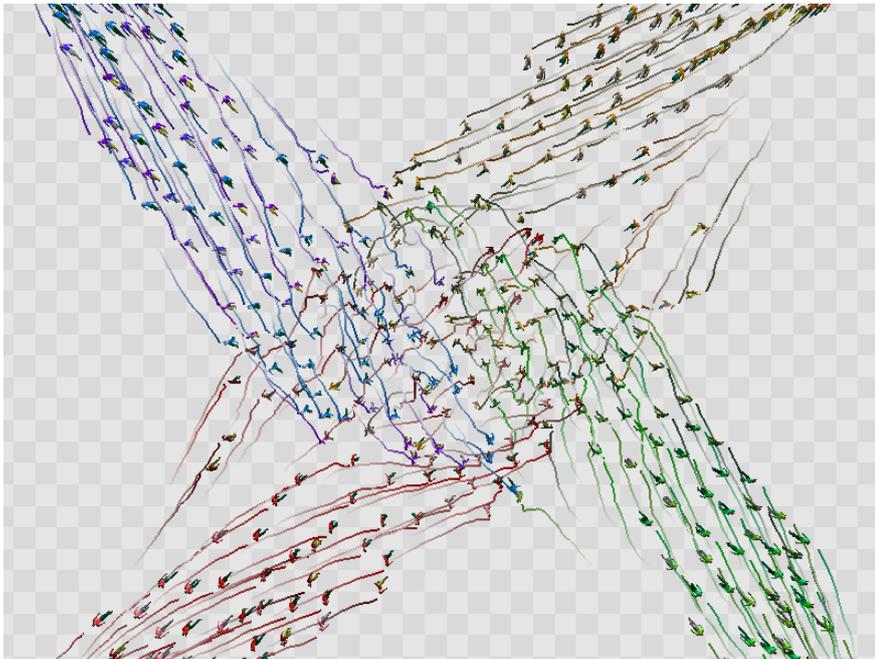


Figure 6.18: An image taken from Treuille et al. (2006) demonstrating their system producing a vortex phenomena.

(2010) are just a selection of such papers. We believe the above quote makes it fairly clear that this is not a common real world phenomena, and as such recreating it has little significant meaning.

This explains why more attempts were not made to provide proof of the realism of our method. Additionally, as previously stated, realism was of a lesser concern in producing our model. Rather the aim was to provide a greater ease of control and creation of scenes which could then be adapted and tweaked by the animator to produce the effects they desire.

6.6 Comparing Reeb Graph to Medial Axis

In this section we will provide a comprehensive comparison of the Reeb graph to the Medial Axis from the viewpoint of finding the topology of a scene for crowd simulation. The Medial Axis is one of the more, if not the most, popular methodology for this purpose. A few examples which use the Medial Axis for crowd simulation are Karamouzas et al. (2009), Pettre et al. (2005) and van Toll et al. (2012b). It therefore makes sense to provide a comparison in order to better justify our decision to use the Reeb Graph.

Given a scene filled with obstacles, the Medial Axis is the set of all points which have more than one closest point on the boundary of those obstacles. This means that it is a graph which accurately describes the topology of the terrain and as such it could potentially take this role in our system which is currently provided by the Reeb Graph. Additionally the edges of the Medial Axis by definition describe the maximum clearance path through any section of the scene. This means that by checking their distance to the surrounding scene they also provide an alternative to the iso-flowlines (as described in Section 4.1) for finding the capacity of each region. These two differences, finding the topology and finding the capacities, are two significant parts of our approach which could be replaced by the Medial Axis. As a result these are the two aspects on which we will focus our comparison.

In the rest of this section we will first compare the topology of the Reeb Graph to that of the Medial Axis in Section 6.6.1, discussing the advantages and disadvantages of each approach with some examples. Then, in Section 6.6.2, we will compare the capacities found using the Medial Axis to those found by our system through the iso-flowlines, this will be done both at the local level of individual capacities and at the global level, examining the effect it has on the maximum flow calculation. Finally, in Section 6.6.3, the conclusions of our comparison will be provided. Throughout the rest of this section where the Medial axis is computed it is done using Fortune's algorithm (Fortune (1987)) with the obstacles treated as solid and the edge of the available area taken as a further obstacle in the space.

6.6.1 Medial Axis Topology Comparison

In this section we will compare the topology of the Reeb Graph to that of the Medial Axis, in the process discussing the strengths and weaknesses of each method. A simple example of both used on a scene with 5 obstacles can be seen in Figure 6.19. For this example a reeb resolution of $R = 5$ was used.

In Figure 6.19 the first thing to note is that both graphs are homeomorphic, that is they register the presence of all of the obstacles as loops in the topology of the graph. Although, as mentioned in Section 3.2.2, the parameters must be chosen for the Reeb graph such that it captures the full topology of the scene, a factor which does not feature in the medial axis. One effect of its generation is that the Reeb Graph is connected to the provided start and end points. This means that for the purposes of path planning the Reeb Graph is a specialised solution, which is more useful for the case where there

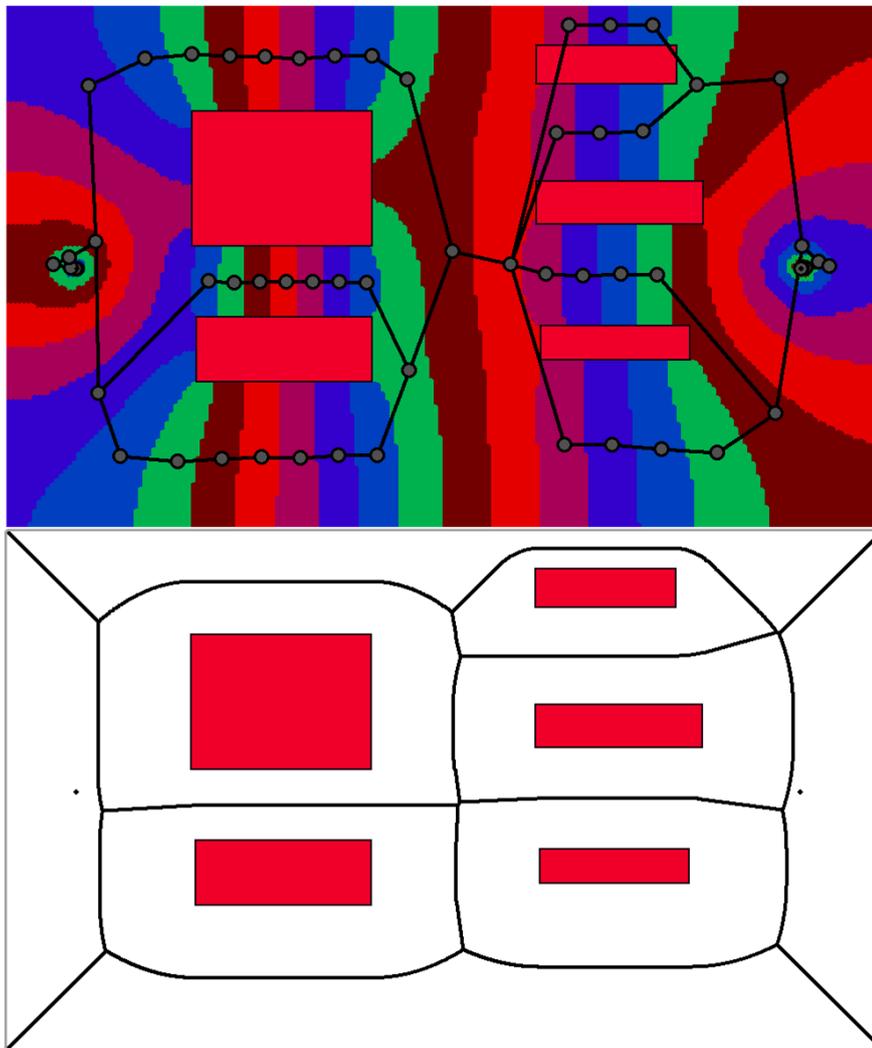


Figure 6.19: The topology graph of a simple scene as found by the Reeb Graph above and the Medial Axis below. The Reeb regions associated with each node in the Reeb Graph are also shown coloured according to their value set.

are a specific set of known sources and sinks between which the crowd is moving. The Medial Axis is a more general solution and it can be connected to given start and end points, however doing so requires it to be further extended.

Though the two graphs are homeomorphic, the Medial Axis is also more complex than the Reeb Graph (in Figure 6.19 the Reeb Graph has 128 edges whereas the Medial Axis has 3551). This will slow down the maximum flow calculation, which is directly bounded by the number of edges, but it is also part of a more complex problem. Within the Medial Axis there are edges which take account of non-convex features of the scene but which, generally, will not describe the navigable topology within it. This can be seen somewhat in the corners of Figure 6.19, however clearer examples are given

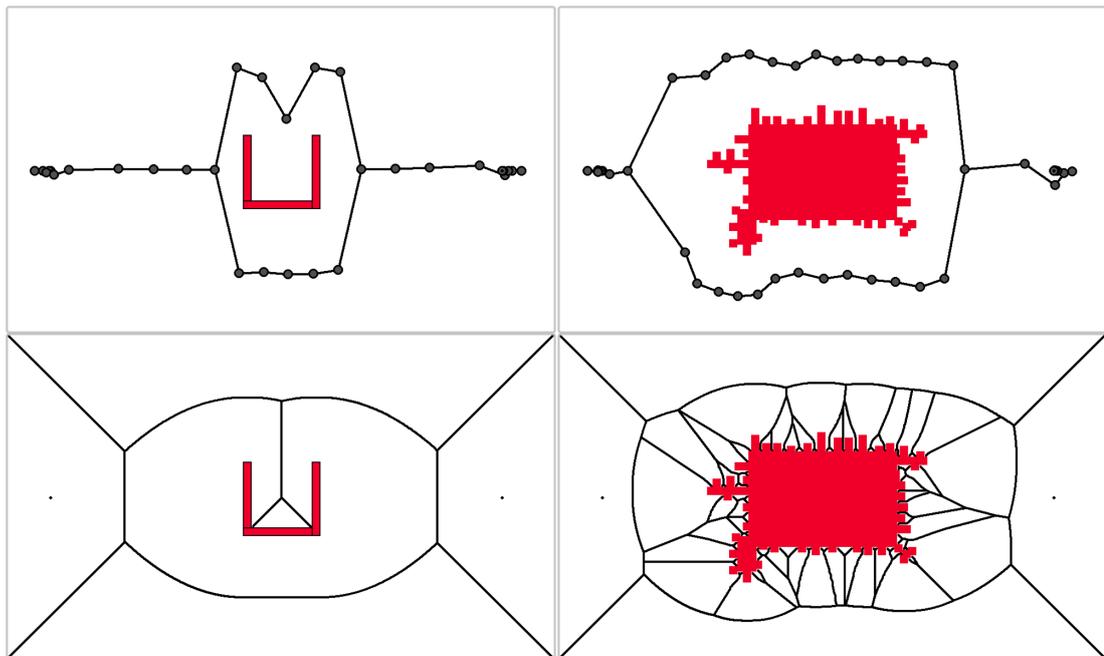


Figure 6.20: The topology graph of two scenes (left and right) given by the Reeb Graph, top, and the Medial Axis, bottom. The left scene has a simple non-convex obstacle in the centre, the right scene is a single obstacle with a bumpy edge. In both cases it can be seen that the Reeb Graph returns the graph of the navigable topology (according to the given start and end points), whereas the Medial Axis includes extra edges.

in Figure 6.20. In this Figure both scenes represent cases where the Reeb Graph is considerably less complex than the graph provided by the Medial Axis. In left example there is a non-convex obstacle in the centre of the scene, which generates extra edges for the inside of the obstacle. In the right example there is merely a single obstacle with a bumpy surface, but each bump on the surface of the obstacle generates at least one extra edge (with a more detailed real life obstacle this problem could be significantly exacerbated). In both cases there are many extra edges which are not required for the problem represented here (moving from the start point on the left to the goal point on the right). Simplifying the Medial Axis to produce only the navigable topology will require extra steps which could result in information about the capacity of the scene being lost. While this could be overcome, it nevertheless represents an extra level of uncertainty to a problem to which the Reeb Graph already represents a good solution.

6.6.2 Medial Axis Capacity Comparison

In this section we will compare the way our method computes the capacity of an open space, as described in Section 4.1, to the same computations done using the Medial Axis. This comparison is important as it represents the other (other than the topology) aspect of our system which could be effectively replaced by the Medial Axis.

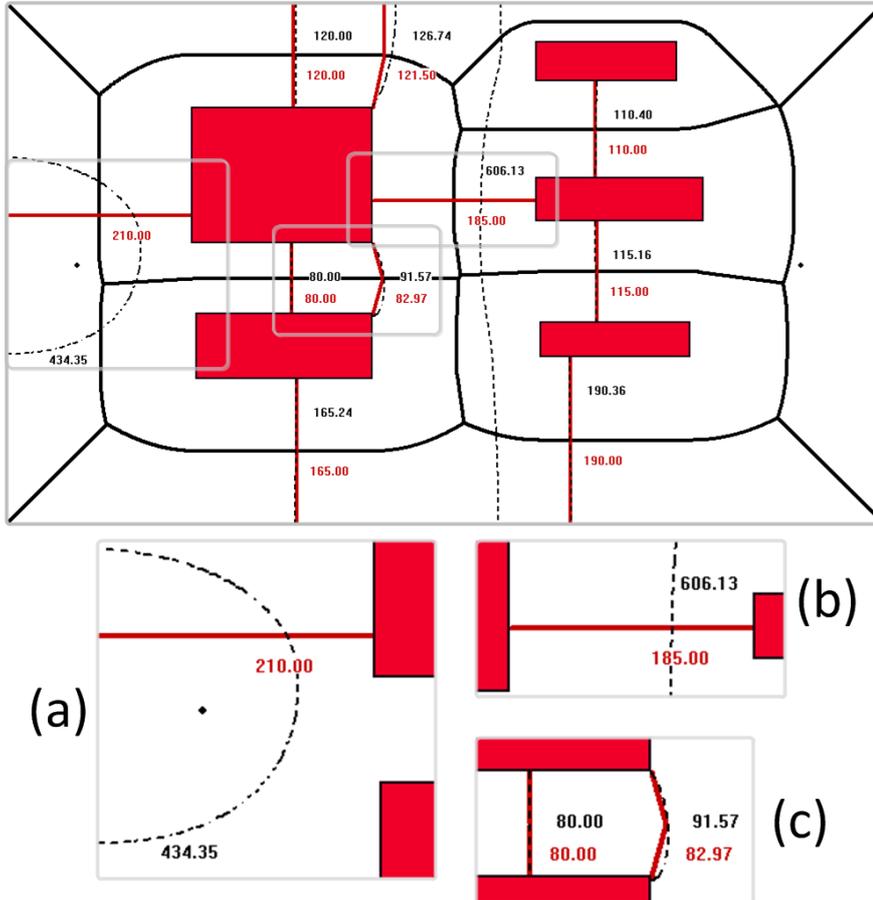


Figure 6.21: The top image shows Medial Axis along with the capacities found at various points by both my system (in black) and the Medial Axis (in red) on the example from Figure 6.19. The bottom three images are close ups of particularly problematic cases, specifically: (a) demonstrates issues with comparison near start or goal points, (b) demonstrates problems with comparison in large open spaces and (c) shows a comparison of values found in a bottleneck of the scene.

A comparison of both methods can be seen in the top image of Figure 6.21. In this figure the width in the scene for a given point x was found using our system by generating an iso-flowline starting from the polygon containing x for the average harmonic field value of that polygon. These iso-flowlines and their lengths are shown in black.

Using the Medial axis the width at x was found first by finding the closest point on the Medial Axis to x , here called y . The closest points on the obstacles or edge of the allowable area to y were then found, resulting in one or more radii defining a circle around y . The width at x was then taken as the diameter of this circle (for illustration purposes only the radii found and the value of the diameter are displayed, both in red). The widths and their associated lines have been shown for a number of different values of x throughout the scene.

In the bottom half of Figure 6.21 three close up examples have been given. The first of these, image (a), demonstrates how the width given by the iso-flowline is different to that provided by the Medial Axis near to the start or goal points. This is because the Harmonic field limits are situated at these points, so naturally the gradient of the field forms a peak or valley centred on them. For our purposes this is desirable, as we consider these to be the areas where agents are emanating from or to and so we want them to be able to arrive or leave from all directions.

The second example in the bottom of Figure 6.21, image (b), shows another place where the results are very different with each method. Here the entirety of the iso-flowline was not included, though it can be seen in the above image spanning the entire width of the scene. The Medial Axis is giving the accurate local width at this point in the scene, however there is an argument in favour of the value provided by the iso-flowline here. Specifically it is that in this scene we know agents will be progressing from the left to the right of the scene and it is providing the width relative to this direction of travel.

The final example in Figure 6.21, image (c), shows the values found at a couple of points at a corridor in the scene. Firstly at the opening of the corridor, on the right, the length of the lines is close but there is some difference. It can be seen that this difference is due to the curve in the iso-flowline at this point which means it is travelling a greater distance and giving a somewhat inaccurate reading about the width of the space here. This is because the harmonic field at this point spreads out as it moves into the open space. Secondly, if we look at the lines deep within the corridor, on the left, the values found for the width are identical. This is because in any bottleneck within the scene the gradient of the harmonic field will be progressing directly parallel to that bottleneck, giving iso-flowlines which provide the exact, or very close to exact, width of that part of the scene (as can be seen in the other examples in bottlenecks in Figure 6.21 above).

Maximum Flow As Figure 6.21 demonstrates the capacities found by both meth-

ods can vary by quite some amount. However, the only reason why we obtain the width of any point within the scene is to find capacities which can be used to compute the maximum flow. For this reason it is important to also compare the maximum flow calculated by using our system to that found using the Medial Axis and the results are for the scene from Figure 6.21 are given below in Table 6.2.

Method used for widths	Maximum Flow found.
Medial Axis	365
RG ($R = 7$)	365
RG ($R = 6$)	365
RG ($R = 5$)	365.099
RG ($R = 4$)	365.2613

Table 6.2: The Maximum flow found for the example in Figure 6.19 by using the Medial axis and various resolutions of Reeb Graph.

For the results in Table 6.2 four different values for the Reeb resolution were used along with the Medial Axis. The results show the validity of using the iso-flowlines to compute the capacities. Although there are potential inaccuracies to the iso-flowline, as shown in Figure 6.21 image (c), in bottlenecks within the scene the values found are normally extremely close or identical to those given by the Medial Axis. For lower Reeb resolutions there is some inaccuracy, however it is well within what we would consider acceptable levels (for all of the examples in Section 6.1 the smallest radius of an agent used is still greater than one, so none of these errors will create any extra agents in the scene). All the same it is important to analyse where this inaccuracy came from. In order to achieve this we performed this comparison for a large number of cases and were able to create Figure 6.22 which represents a worst case for our system.

In Figure 6.22 the Reeb Graph displayed has a resolution of 4 and the iso-flowlines shown are the shortest found for each Reeb Edge (that is, they are the ones used to determine the capacity). The issue here is that because the obstacles are so thin relative to the direction of the gradient of the harmonic field the iso-flowlines drawn in the bottleneck are more curved. As shown in Figure 6.21 image (c), near the edge of a bottleneck the gradient becomes curved and less accurate and with a thinner obstacle the majority of space in the bottleneck is near the edge.

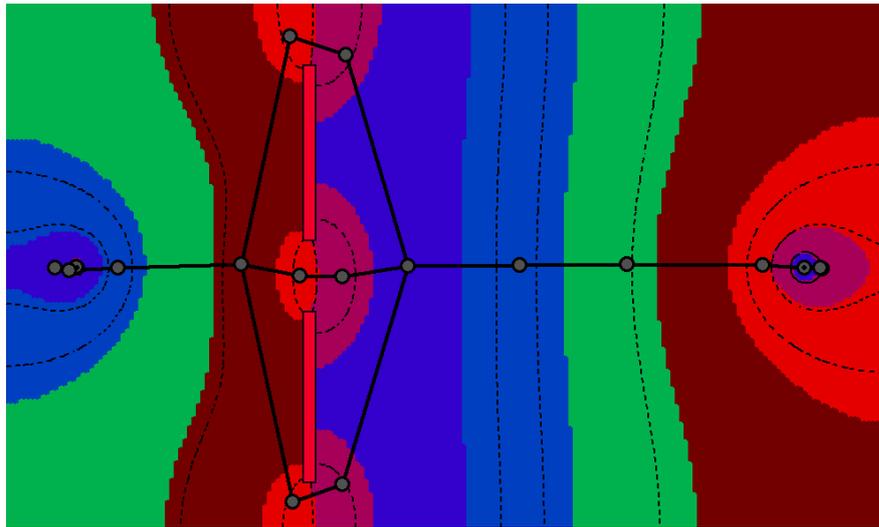


Figure 6.22: A simple example of the worst case for our system with the Reeb Graph displayed, also shown is the shortest iso-flowline found for each edge. It can be seen that for the gaps between some of these obstacles the iso-flowlines are curving (thus will report slightly more than the actual width at this point).

The results for calculating the maximum flow on Figure 6.22 are shown in Table 6.3. Here it can be seen that our system performs worse than in the previous example due to the more difficult thinner obstacles. For the lower resolutions there is an error of greater than 1 which is high, but still not particularly large given that this is the worst case for our system. One of the reasons for this is that there will be more edges in the Reeb graph at bottlenecks, as these are the points where the harmonic field's gradient is at its steepest (as can be seen in Figure 6.22 the reeb regions are a lot thinner and closer together at the bottlenecks). In spite of this we do recommend that users should be aware of the potential for the maximum flow to be too high in cases where obstacles are very thin relative to the direction of flow of the harmonic field.

6.6.3 Medial Axis Comparison Conclusion

For the purposes required by our system the Reeb Graph and Medial Axis are, in the main areas, interchangeable. This can be seen in the topology as we have demonstrated that the Reeb Graph and Medial Axis are homeomorphic. We have also shown that though there are inaccuracies at very low Reeb Graph resolutions, the iso-flowline capacity calculations are equal to those performed using the width of the Medial Axis for the specific purposes of computing the maximum flow.

In order to use the Medial Axis within our approach additional extension would

Method used for widths	Maximum Flow found.
Medial Axis	200
RG ($R = 9$)	200
RG ($R = 8$)	200.099
RG ($R = 7$)	200.1231
RG ($R = 6$)	200.6698
RG ($R = 5$)	201.7343
RG ($R = 4$)	202.5551

Table 6.3: The Maximum flow found for the example in Figure 6.22 by using the Medial axis and various resolutions of Reeb Graph.

be required. This is true in terms of adapting the graph to the specific topology of a start and end point. It is also true in terms of providing pathing information to the agents. The pathfinding with our system is provided by the Harmonic field which is computed as part of the Reeb Graph, in the form of flowlines. However, entire papers have been dedicated to pathfinding with the Medial Axis, such as in Geraerts (2010), and these do not do so in a manner which allows providing paths for groups of agents while avoiding them obstructing one another at bottlenecks.

In conclusion our method is simply better tuned to this specific problem. There are some small accuracy gains which could be had from using the Medial Axis, but they are outweighed by the implementation problems and difficulties which would come with using a more general methodology.

Chapter 7

Additional Applications

In addition to producing a controller for crowd scenes, we also applied the ideas behind the system to two further problems. In this chapter we describe these experiments. Firstly, we used the system to simulate evacuation routes as a tool for designing buildings, this is described in Section 7.1. Secondly, we created a controller for the motion of teams of players in team sports by adapting the ideas behind the system, this is described in Section 7.3. These experiments were intended primarily to show that there was potential to develop the system in these areas and to demonstrate that the system presented generalises to a broader class of multi-agent control problems.

7.1 Evacuation Design

In this section we explain how we applied our system to the problem of designing buildings with evacuation in mind. Our system generates a series of agent paths, and also estimates how many agents will move along each path (this is described in Section 1.2.1, with equations (1.1) and (1.2) both repeated below for clarity).

$$X_{paths} = f(scene, Y_{agents}, starts, goals) \quad (7.1)$$

$$Y_{agents} = g(scene, goals) \quad (7.2)$$

By combining the information which we have provided in the form of these X_{paths} and Y_{agents} we already have useful information about how the layout of buildings will shape the behaviour of pedestrians. By providing this information in a structured way the system can be applied to both the design of new buildings and the adaptation of existing buildings.

There are two ways of using our system in this area. Firstly, the X_{paths} can be extracted to provide immediate visual feedback about the optimal evacuation paths through a given structure. Additionally, further information can be gathered from the Y_{agents} numbers. This is useful for evaluation of the building and to suggest modifications, this process is described below in Section 7.2.1. Secondly, the shape of the X_{paths} can be analysed to provide information about where exit signs are most needed, this process is explained in Section 7.2.2.

As previously explained in Section 6.1 our system is applied to evacuation situations by setting up each building as a series of sources, where people are originating, and sinks, at the exits from the building. An example such system was given by Figure 6.3. Such a representation assumes that agents within the building arrive from common points of origin rather than any haphazard location. This models staircases arriving on the ground floor or central lecture rooms, and we believe that such evacuation situations are ubiquitous enough (as in any multi-floor building) that this is not an issue.

7.2 Related Work

In this section I will give a brief overview of the previous work done in evacuation simulation with details about how it relates to this method. Often these are not research but commercial applications, with the methods consisting of a large integrated set of different systems, however, I will attempt to present the essential idea behind each method.

The most common approach by far is agent based systems where each agent in the crowd is modelled individually with their own attributes, decision making and knowledge about the world. For instance in Lee et al. (2010) they use the BDI framework (originally presented in Rao and Georgeff (1991)) which models agents as a set of beliefs, desires and intentions, with each contributing to the agent's decisions. There are also much more detailed systems such as the EXODUS model, Gwynne et al. (2005), which model agent attributes such as age, breathing rate and running speed, as well as adapting these in reaction to environmental aspects, such as toxic gases. The EXODUS model bears many similarities to our own, as it uses an abstract network of nodes and edges to represent the given building. However, as their system is agent based, the agents move around this network of their own accord, progressing towards the exits based on their individual attributes and reactions to the environment. Agent models

are used primarily because they allow the modelling of different types of environmental factors in modelling situations such as evacuees with disabilities as in Christensen and Sasaki (2008) and evacuating from a building through which fire is spreading as in Shi et al. (2009). One big disadvantage of agent based approaches is that they are very computationally expensive, which means it is difficult for them to provide a visual representation of what is happening within a building. The main difference between our method and these agent based approaches is their lack of global information. This means that we are able to identify overall features in the scene such as the likely bottlenecks, whereas they are better able to model aspects such as panic within crowds.

Another class of approach in evacuation modelling is cellular automata, where the scene is abstracted away to a grid where each square is either occupied or not and agents move based on some weighting into a neighbouring cell at each timestep. There are a few examples of this type of model, SGEM (Lo et al. (2004)), PedGo (Klüpfel and Meyer-König (2003)) and EGRESS (Ketchell et al. (1993)). They differ in complexity, from simply providing a list of exit routes, to providing full simulation with simple behavioural attributes for the agents, but they all work on some sort of field propagated over the grid from the exits. The main advantage of these approaches is that they are fast, because they abstract much of the detail of the system. This means that they can even provide visual representations of the simulation as it is occurring. Their primary drawback is that because they discretise the space a lot of information is lost and this can lead to inaccuracies.

The most important comparison for our method is EVACNET+ (Kisko and Francis (1985)). There are many similarities between this method and ours. They are also a network based method, which again represents a building as a series of nodes (rooms) and edges (corridors/doors), so they solve a similar flow based problem. The features they offer are also very similar to our method, such as finding optimal evacuation plans, identifying bottlenecks and giving floor clearing times. There are a few advantages which our system has over theirs. Firstly, they require that a building be entered manually as a series of nodes and edges, with the values given for each, while this is a process which we wholly automate. Secondly, they provide no visual representation of the evacuation in action, as they do not have agents, only a set of numbers to represent the situation of the building. Finally, although they are able to offer information to users, they are not able to do so interactively. With our system a user can change the structure of a building and see within a few seconds how that alters bottlenecks within it (as described in Section 7.2.3).

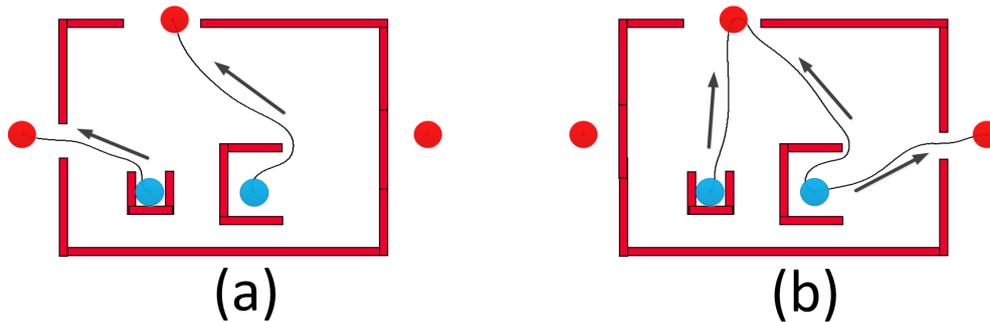


Figure 7.1: The evacuation routes found by our system for two example exit configurations, the sources are marked with the blue circles and the potential sinks with the red circles. (b) is less desirable as the agents from the larger staircase have to split up to fully utilise the exits, something which would be difficult to enforce in practice.

There is much we can learn from looking at other evacuation systems. For instance for future development of our evacuation approach it would certainly be useful to allow the system to take CAD drawings of buildings as inputs (something many of the above methods do). While our method is not as comprehensive as some of the large commercial systems presented above, it contributes some useful new insights.

7.2.1 Pathing Analysis

In this section we demonstrate how our system provides immediate feedback to a designer on the paths which agents will follow when leaving a building. This is important as it provides information on where emergency exits should be placed expressly to avoid certain undesirable situations, such as large groups of agents all heading towards too small an exit. It is possible to see the paths of all agents based on a sample run of our system, simply watching what paths they take from step to step, and to get a better idea of the overall manner in which the agents are utilising the environment. This is a useful approach, however it is time consuming and potentially difficult to visualise. For the pathing analysis explained here it is much clearer to display the shortest path provided to the agents along each route as shown in Figure 7.1.

In Figure 7.1 a very simple evacuation scene is shown for demonstration. Here there are two sources, representing a large and a small staircase in the centre of the building, there are also three sinks set up to represent the potential exits from the building. The same number of agents can evacuate in both case (a) and (b), but in case (b) there are two routes from the larger stairwell, one leading upwards and one to

the rightmost entrance. This split happens because the agents from the larger staircase cannot all leave via the top exit, as they must also accommodate agents from the smaller staircase. This split is clearly undesirable, as it requires that the group of evacuees who reach the bottom of the large stairwell must make a decision to split up and travel in two different directions. In a real world scenario even assuming that some sort of barrier is used to direct the crowd, such a perfect split would be difficult to attain. In case (a) it can be seen that by switching the secondary exit from the right to the left hand side of the building, this provides a route for the evacuees from the smaller staircase and allows those from the larger staircase to all take the main exit at the top.

As well as providing feedback about the paths of individual agents, we also provide further information at the request of the user based on analysis of these paths and the Y_{agents} numbers. Specifically, we can inform the user about the length of the shortest path associated with each route out of the building, along with the average path. We can furthermore state how many people evacuate the system in the maximum flow case for each different configuration. Finally we can also indicate the Minimum cut of the graph. The Minimum cut is the set of edges which can be cut to segment the Reeb graph into two while cutting through the least amount of capacity (that amount of capacity will be equal to the Maximum flow). It provides an immediate indication, even in complex scenes, of where the bottlenecks are and thus where to expand the scene to allow an increased flow.

This type of analysis of different building configurations can be performed in a matter of seconds with our model, and provides useful feedback about the optimal evacuation routes. It is a simple and effective extension of the pre-existing methods of our system which could quickly and easily be used to make decisions about building safety such as the placement of emergency exits.

7.2.2 Signage Analysis

In this section we explain how our system can be used to assist in placing emergency exits in a building. This must be done to give residents proper information about which exit they ought to be proceeding to in order to safely evacuate. In NFPA (2006), the fire safety guidebook put together by the National Fire Protection Association of the United States, they state:

7.10.2 Directional Signs. A sign complying with 7.10.3 with a directional indicator showing the direction of travel shall be placed in every

location where the direction of travel to reach the nearest exit is not apparent.

Often the required direction of travel will be simple to determine, e.g. a staircase will terminate at an emergency exit. However, all too often with large empty lobbies or long corridors with many turns it is non-trivial to determine not only which exits people should be heading towards but also at what point in their path this will no longer be apparent to them (and thus need signs directing them). Failure to properly direct people in such situations may cause significant issues, such as too many evacuees heading to one single exit, causing dangerous delays and congestion. In this section we explain how, by analysing the curvature of the paths provided by our system, it is possible to take some significant steps towards automating the placement of these signs.

We make the assumption that places where agents are making significant turns in their paths out of a building will also be places where their next direction of travel may not be apparent to them (and thus where signs are needed). Therefore in order to mark the areas which require signage, each path out of the building is analysed in turn to find the points where these paths have the highest curvature, that is, where the agents will be altering their direction. For each route the shortest path is chosen. These paths are made up of flowlines through the environment as described in Section 3.1.4 and they consist of a list of points describing the path for the agents to follow. To find the areas of highest curvature the list is stepped through starting from the i^{th} point on the list, where i is initialised to s and s is a value determining the size of the steps we are taking through the list. The curvature value C for a given i is found using the equation:

$$C_i = \frac{\mathbf{V}_{-i} \cdot \mathbf{V}_{+i}}{\|\mathbf{V}_{-i}\| \|\mathbf{V}_{+i}\|}, \quad (7.3)$$

where \mathbf{V}_{-i} is the vector between the i^{th} point on the path and the $(i-s)^{th}$ point and \mathbf{V}_{+i} is the vector between the i^{th} point and the $(i+s)^{th}$ point. Here it can be seen that s is the resolution at which curves are being examined, which allows the system to ignore very small curves. The list is stepped through in this way in increments of s and at each point i there is said to be a large enough curve if the following condition is satisfied:

$$C_i + C_{i+s} > T, \quad (7.4)$$

where T is a threshold value set to ensure that the curvature of two adjacent sections of the path C_i and C_{i+s} must both be high. Subsequent values for i which also have a value greater than T are considered to be a part of the same curve and they are grouped

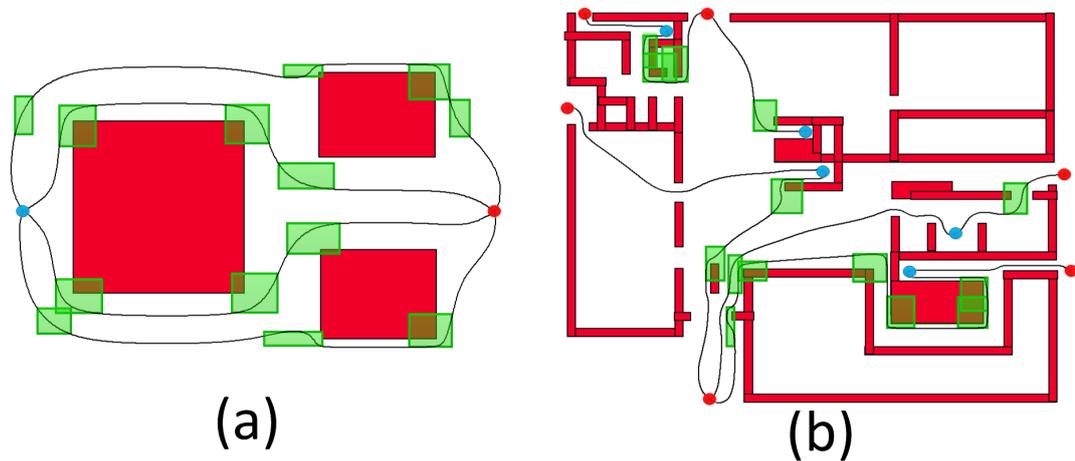


Figure 7.2: Two example scenes with the primary path for each route displayed and the points of high curvature, found by our system, highlighted along each of these by the green boxes. In each scene the sources are marked by the blue circles and the sinks (exits) by the red circles.

together. Once a group is complete a boundary is drawn around them representing the area where agents are turning in their paths. These boundaries are recorded for every path for easy display to the user.

An example of the output from this algorithm is shown in Figure 7.2, where image (a) is a simulated environment and the scene in image (b) is based on the floor plan of a real building. Here it can be seen that areas where agents are undergoing a large change of direction are highlighted, indicating that they are prime candidates for sign display. It is worth pointing out that turns which happen over a very large region may go undetected, such as the bottom left of Figure 7.2 (a). This is dependent partially on the value for the threshold T , but for large curves such as this it is more reliant upon the step size s . Additionally, in both examples in Figure 7.2 there are a number of smaller curves which can be seen near the start and end points which have not been marked by our system. These are being intentionally ignored, as in complex scenes such as these the curvature of the flowlines is nearly always extremely high near to the start and end points. As a result marking these would cause a great number of false positives.

7.2.3 Evacuation System Experiments

We performed a number of experiments to demonstrate the effectiveness of these methods. Firstly, as discussed in Section 7.2.1 we performed some experiments providing feedback on the Minimum cut and Maximum flow of a scene, these can be seen in

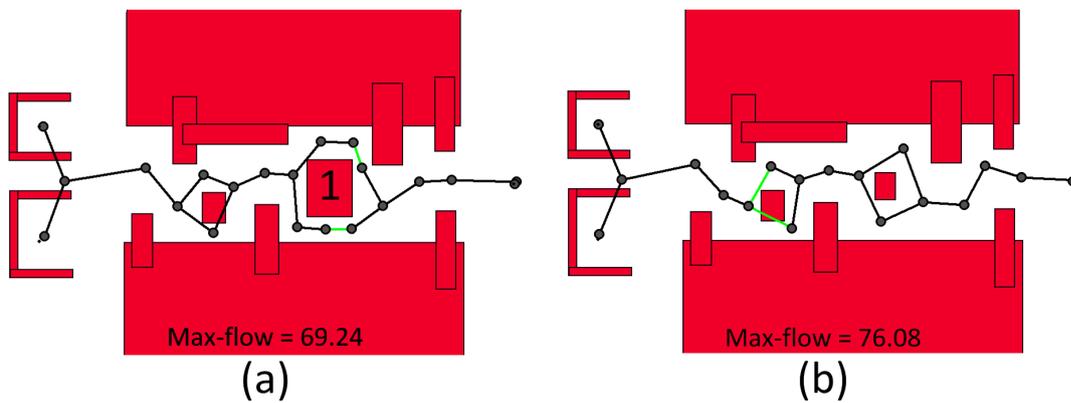


Figure 7.3: Here both images show a scene with two entrances on the left and one exit on the right, with the Reeb graph and maximum flow displayed. The scene is complex and it is difficult to locate the bottlenecks. Image (a) shows the min-cut edges highlighted in green. Image (b) shows the system after improvements are made to avoid that bottleneck, with a new max-flow and min-cut.

Figure 7.3. Here both images show an evacuation scene with two sources and a single sink separated by a complex corridor filled with obstacles. It is difficult to determine where the Minimum cut is along this corridor. Our system is able to inform the user of the Maximum flow through this scene and identify the Minimum cut in the scene, as shown in image (a). Once the Minimum cut is highlighted it can be seen that shrinking the obstacle marked '1' will increase the flow, as shown in (b). This demonstrates how easily our system can be used to find and avoid bottlenecks even in complex scenes.

Secondly we further demonstrate the effectiveness of our signage analysis presented in Section 7.2.2. Figure 7.4 shows another large building evacuation scene, this time with one lecture theatre also occupied implying an internal source. Here again it can be seen that what we would expect to be the main decision points have been highlighted. To further verify the validity of this method we also compared the results of our system from Figure 7.2 image (b) to the signs found in the building upon which it was based and the result of this can be seen in Figure 7.5. This is a good match for our system, with many of the actual exit signs placed near, or within line of sight of, our indicated placements points. This indicates strongly that our method is a valid metric for indicating the placement of signs throughout a building.

There are several signage points which can be seen through Figure 7.5 (a) to exist in the real world, but which our system does not currently replicate. Firstly, along corridors the actual building has signs placed wherever there is a door such that in any section of the building the required exit is clearly indicated. By placing doors in our

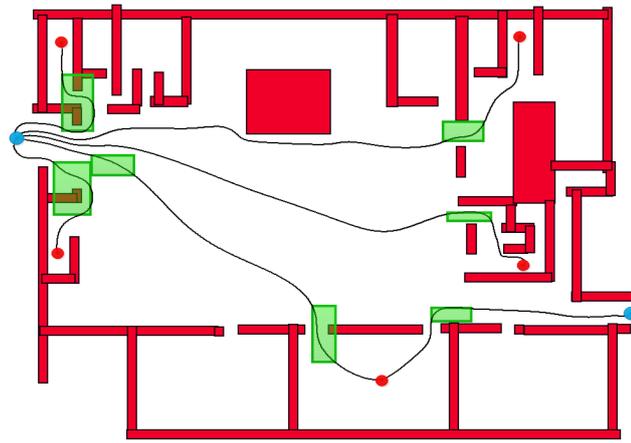


Figure 7.4: An additional example evacuation scene with two exits from the building and 5 sources within it, again taken from the floor plan of a real building. The points of high curvature along the paths are marked with the green boxes, the sources are marked with blue circles and the sinks are marked with the red.

system it would be possible to replicate this effect simply by looking for where any route crossed a door. Secondly, in large open spaces the exit signs are placed by the exit and are specifically designed to account for line of sight. This can be seen at the two positions in the centre marked as '1' and '2' in Figure 7.5 (a), here '1' is on a wall such that it can be seen by agents approaching from the right, and '2' is placed to be seen from above. We believe it would be useful to incorporate some representation of line of sight into our system to take account of large open spaces in such cases. In this case simulating the line of sight is relatively simple, since the paths we have give us a direction of travel from one node to the next we know which direction the agents will be facing. Given a normal field of view (humans have a a large frontal field of view) we can then trace this outwards from any decision point to nearby walls. Further simulation could involve restricting this field of view to a certain distance, to simulate the conditions in a smoky building.

Our system is best suited to problems such as these shown here, where there are a specific set of sources where agents will enter the scene and another set of sinks where they will leave, as in the ground floor of a high rise building. It is less well suited to evacuation scenarios where the agents are pre-existing, already spread throughout the scene, as in the stands of a football stadium. Here the same method could be applied but it would require that the scene be abstracted somewhat, with each large group of agents (each section of seats in the stands) combined and treated as a single source

node in the graph. Simulation within one of these combined nodes could be done by a separate method, with our approach dealing with the flow exiting from each such node.

7.2.4 Evacuation System Conclusion

In summary, we have provided extended our system to provide a surprising depth of information about a building and its structure and it does so quickly and with no additional input. We examine the optimal situation as found by our maximum flow calculation and this means that we are simulating an ideal evacuation, clearly there are useful insights to be found from studying such situations. In future we would like to extend this work to provide further automation. Specifically, it should be possible to give warnings automatically about undesirable situations, such as the agents splitting up, as discussed in Figure 7.1. Also extending the agent controller to provide some representation of panic could allow us to provide a greater depth of information.

7.3 Football Controller

Additionally, we explored how the concepts presented in this thesis could be adapted and applied to the control of cooperating groups of agents in team sports. Specifically, we wish to control the movements of a team of football agents in avoiding their attackers and successfully manoeuvring the ball to the goal. Unlike the evacuation design this is quite a different problem to our main system. Specifically, we are looking to find the function h as defined:

$$T_{t+1} = h(T_t, O_t, B_t), \quad (7.5)$$

where T_t is the full set of positions of the team we are controlling at time t , O_t is the positions of the opposition team at time t and B_t is the position of the ball or puck. Here h should be structured to optimise the T_{t+1} in order to most improve that team's chances of winning the game. The fundamental difference from our main system is that in this scenario we only try to find the best next move for the agents on the controlled team, rather than planning out the entire motion. This is because the future positions of the opposition team are unknown and difficult to plan for without a full model of their behaviour. Nevertheless many of the same principles from our main system still apply. Specifically, we drew a graph which describes the current state and applied the maximum flow calculation to this graph, in order to evaluate the current state. The h function is then based upon this evaluation of the current state.

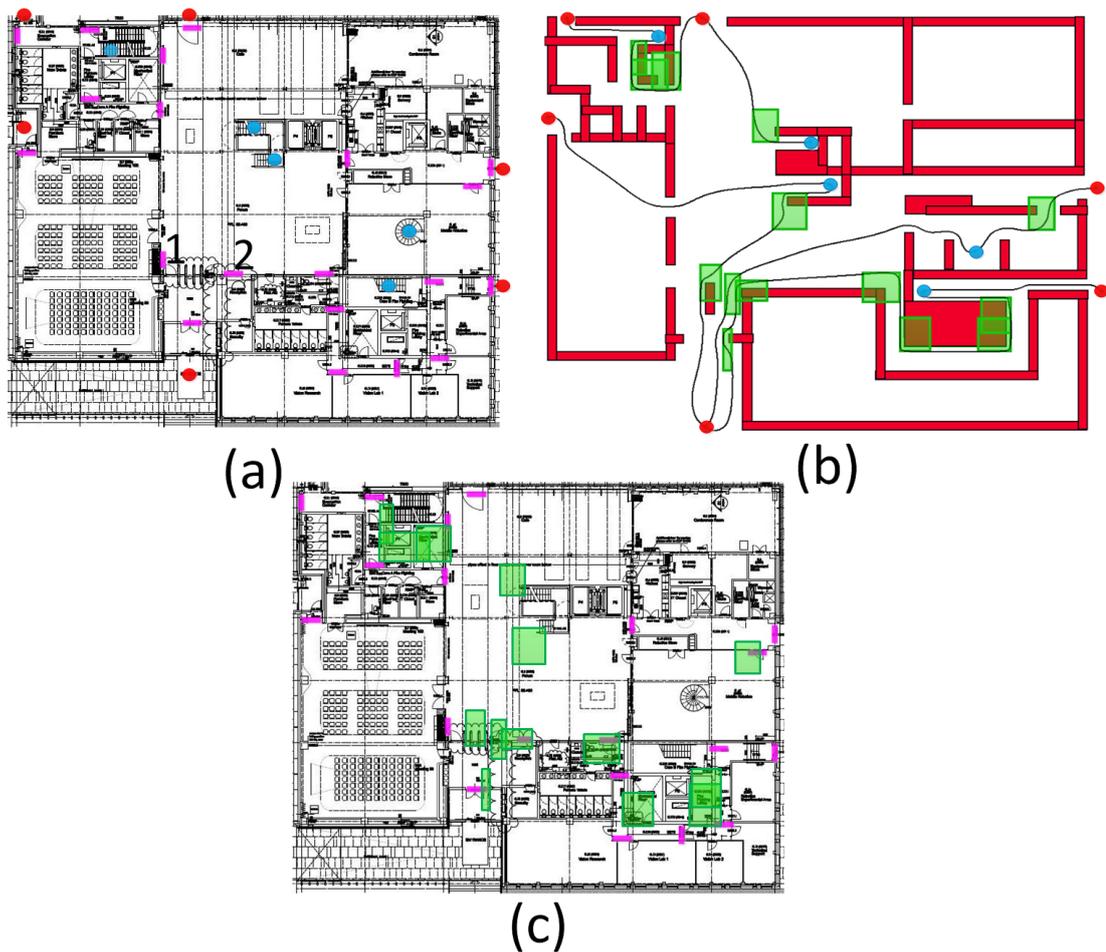


Figure 7.5: Here (a) shows a building blueprint with exits marked in red, points of arrival on the ground floor marked in blue and emergency exit signs marked in magenta. The signage suggested by running our system on this building shape are shown in (b). The suggested signage is overlaid on the blueprint in (c). It can be seen that in tight spaces there is a close match between our suggested placement and the actual signs. In open spaces the exit signs are instead placed within clear line of sight of the regions marked by our system.

This approach could be easily applied to a number of team games, in the work here it was used to provide control for the attacking team, as T , in a game of football, when they are in possession of the ball, with the opposing team O being the defenders. Controlling the attacking team involves not just deciding where to move, but also when to pass and shoot in order to get the ball into the goal. The solution could be just as well applied to the problem of defending against such an attack, but for demonstration purposes we wrote our own basic defender controller. The system operates primarily by providing an evaluation of the current state of the game in any given situation and this is described in Section 7.3.1. We then explain how this evaluation is used by the system to provide control for the agents and what additional rules were used to provide control for shooting and passing situations, this is in Section 7.3.2. Next in Section 7.3.3 we explain the control which we produced for the defending players. In Section 7.3.4 we present our experiments and evaluation of this system. Finally in Section 7.3.5 we present our conclusions along with discussion current limitations and how they will be overcome.

7.3.1 Evaluating the Game State

In this section we explain how the game state is evaluated. This is a three step process: first a graph is built which represents the possibilities in the game, then the edges of the graph are evaluated, then the graph as a whole is evaluated based on the values of those edges. The intuition of this whole process is that team games can be modelled as graphs, where each of the edges represents a potential pass or shot at goal, and the value of the state is represented by the maximum flow from the player with the ball, as a source, through passes and shots to the goal, as the sink. This gives an evaluation of the game state which takes into account all current possible routes from the ball to the goal. This provides an excellent metric for evaluating the entire game state at any point in time which naturally returns that evaluation as an easily usable single number.

The first stage in performing this evaluation is to create the graph. This is done by adding an edge between every pair of attacking players, and between every attacking player and the centre of the goal. In total this means there will be $\frac{n(n-1)}{2}$ edges between the n players, plus n edges to the goal. We found this size of graph to be easily dealt with, especially as there is a maximum of 11 attackers. An example such graph can be seen in Figure 7.6.

Once the edges are in place they are evaluated by considering the length of the edge

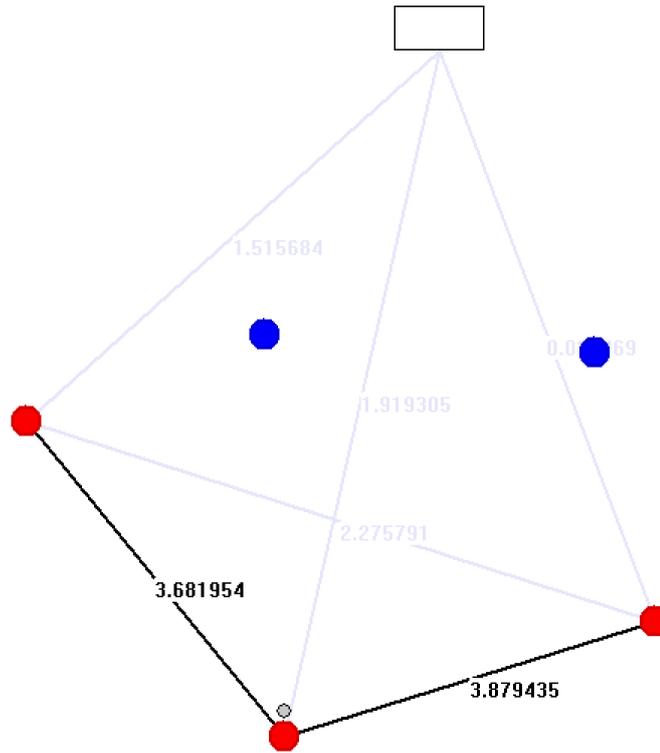


Figure 7.6: An example graph of a football game produced by our system. Here the attacking players are in red, the defending players are in blue, the goal is at the top and the player at the bottom can be seen to be in possession of the ball. The edges are coloured according to their value, with edges with a value greater than a threshold p shown in black and those below p shown in grey (see Section 7.3.2 for an explanation of p).

and the distance from the defenders to the edge. This is done using the equation below:

$$E_i = \frac{\eta}{l_i} \left(\prod_d D_{di} \right)^\kappa, \quad (7.6)$$

here the value of each edge is given by E_i , the length of those edges is given by l_i and the interference value for each defender d on the edge i is given by D_{di} . Additionally, η and κ are values used to determine the relative importance of the length of passes and the interference of defenders respectively, and the values chosen are explained below. The interference value itself is found using the equation below:

$$D_{di} = \frac{1}{1 + e^{\frac{30 - X_{dj}}{10}}}, \quad (7.7)$$

this models a Gaussian influence for each defender on every pass, with a much higher influence at closer distances. Here X_{dj} is the distance of a given defender dj from the

edge i . It also allows multiple defenders to contribute towards blocking the same pass. The parameters in Equations (7.6) and (7.7) were chosen because they were found experimentally to replicate desired effects from actual games. In our experiments we chose $\eta = 1000$ and $\kappa = 5$. These values were chosen to give defenders close to a pass a high influence over it, defenders far away very little influence and passes without interference a large value unless they are very long.

Finally we extend these equations to include shots at goal. The value of an edge between a player and the goal is computed as follows:

$$E_i^{Shot} = E_i \cos(\sigma), \quad (7.8)$$

with E_i evaluated as in Equation (7.6). Here σ is the angle between the shot and the vertical, indicating the fact that shots from directly in front of the goal are quite easy, but shots from oblique angles off to the side have a smaller target and are much more difficult. Every edge in the graph is evaluated through these equations and examples can be seen in Figure 7.6.

Having created the graph and evaluated each of the edges all that remains is to compute a value for the current game state. As mentioned this is done by using the maximum flow computed over the graph, with the player with the ball as the source and the goal as the sink. This process takes into consideration all possible routes to the goal simultaneously. The actual calculation of the maximum flow is done by the same method from Boykov and Kolmogorov (2004) as described in Section 4.2.

7.3.2 Controlling the Attackers

In this section we explain how the value of the game state, described in Section 7.3.1, is used to control the attacking agents. Initially this was done with the finite differences method, that is by varying the position of each agent and then re-applying the evaluation of the game state to see how it was effected, as shown in Figure 7.7 (a). This allowed us to construct a Jacobian describing the gradient for the current state over which we could optimise, to find the best combination of movements for the set of agents. Although this evaluation metric looks at the team position as a whole, agents tended to move individually towards local minima of their own spaces, which is undesirable in a team based game and precludes any motion which represents cooperation.

As a result we decided to abstract away the positions of all of the team by instead looking at the formation of the attackers as a whole and varying the shape of this

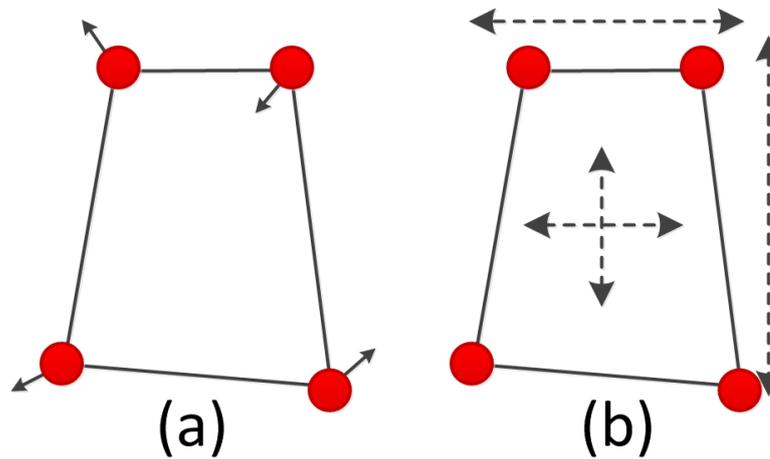


Figure 7.7: An example demonstrating the individual control system for the attackers (a) and the formational control (b). In (b) the dashed arrows represent the 4 variants of formational control used, specifically: moving the entire formation along the x and the y axis, or expanding/shrinking the formation, also along the x and the y axis.

formation. This is an especially useful abstraction as most team sports use formations as a means to control the positions of players on the field. Formations were shown to have a significant effect on the outcome of passes within actual games by Bradley et al. (2011) and they have also been used for evaluation of game state in Robocup tournaments e.g. Almeida et al. (2009).

In order to control the team through the formation we first allow expansion or shrinking of the formation in both the x and y direction by finding the average position of the agents and then moving them all towards or away from that position. We also allow translation of the entire formation, moving every agent in both the x and y direction without changing their position relative to one another. Both processes can be seen in Figure 7.7. Taking such a representation locks down the attackers into their current formation to some extent and this can cause issues in limiting formation changes, but a considerable range of motion can be produced from just these two simple alterations to the formation. By using the same representation it would also be possible to add a much greater range of movement by including other formation based movements such as rotating around a single point or skewing the formation around a single edge. Most significantly, using formational control in conjunction with the maximum flow evaluation allows the players to make moves which are worse for them individually but which improve the situation for the entire team.

The control of the attackers is then performed by finding the gradient caused by

moving a small amount in the positive and negative x and y directions for both expansion of the crowd and translation of the crowd. The next motion can then be found by computing this gradient and moving in the optimal direction for both formational movements. The pseudo-code for this method can be found below in Algorithm 3.

Data: CurrentAttackerPositions, FormationChanges

Result: An updated CurrentAttackerPositions

Movements = $[-n, 0, n]$;

ChangesToMake = [];

foreach Change *in* FormationChanges **do**

 Maximum = EvaluatePosition (CurrentAttackerPositions);

 BestMove = Null;

foreach Move *in* Movements **do**

 TempPosition = ChangeFormation (Change, Move,
 CurrentAttackerPositions);

if EvaluatePosition (TempPosition) \geq Maximum **then**

 Maximum = EvaluatePosition (TempPosition);

 BestMove = Move;

end

end

 ChangesToMake \leftarrow (Change, BestMove)

end

newPosition = CurrentAttackerPositions;

foreach (Change, Move) *in* ChangesToMake **do**

 newPosition = ChangeFormation (Change, Move, newPosition);

end

CurrentAttackerPositions = newPosition;

Algorithm 3: The algorithm which updates the positions of the attackers.

In Algorithm 3 we show how all of the attacker positions are updated. The input to the algorithm is the current attacker positions and the formation changes (those demonstrated in image (b) of Figure 7.7, expanding, shrinking and moving in both the x and y directions). The formation changes are applied in turn with the `ChangeFormation` which takes as input a type of formational change, an amount to change by for that change and a set of positions for the attackers. The amounts to change the formation by are taken from `Movements` which consists of small changes of degree n (here n

is set to $1/5$ the width of a player). After applying each change it is evaluated using `EvaluatePosition` which uses the maximum flow to evaluate the game state as described in Section 7.3.1. The best move from `Movements` for each type of formation change are recorded then these are all applied to produce a new position which is then applied to the attackers.

This controls the movement of the attackers, but for a full game simulation passing and shooting behaviour is also needed, we did this with a simple hierarchy of rules which determine when passing or shooting should happen. Passes occur in two situations, firstly, the player with the ball looks at all available passes and considers those which have a value greater than some threshold p as being valid. Each valid pass is then evaluated by simulating switching control of the ball to the player at the other end of that pass and re-evaluating the game state in that situation. If any of the valid passes would result in an improvement of the game state then the one which will cause the biggest improvement is chosen and taken. Secondly, if there are valid passes but none which improve the game state are available then a clear path to goal is also looked for. That is, a series of passes and shots, starting from the player with the ball and ending at the goal, all of which are above the threshold p . The reason why these are needed is that because the entire game state is being evaluated all at once, there are rare situations where a pass may improve the ball's chances of reaching the goal without raising the value of the game. Passing along such clear path is a likely successful route to score a goal and as a result when one is found then the first pass in that path is taken. Shots at goal occur simply whenever they are available and above the threshold.

When a pass is occurring, in order to avoid agents changing direction and moving out of the path of the ball being passed to them, the evaluation is put on hold and instead the entire formation moves along the same gradient as it chose the last time an evaluation occurred. This ensures that movements can be predicted for the purposes of passing. This is not an issue as passes generally take less than 3 time steps as they are, as might be expected, quite fast.

7.3.3 Controlling the Defenders

As previously mentioned football is a dynamic game so playing without defenders or with static defenders provides no clear indication of the validity of the method. For this reason we also had to produce controllers for the defence. It would be possible to use the same evaluation metric to control the defenders, that is, control them to minimise

the maximum flow from the ball to the goal, but we felt that a simpler controller would be more suitable for a preliminary exploration of the method.

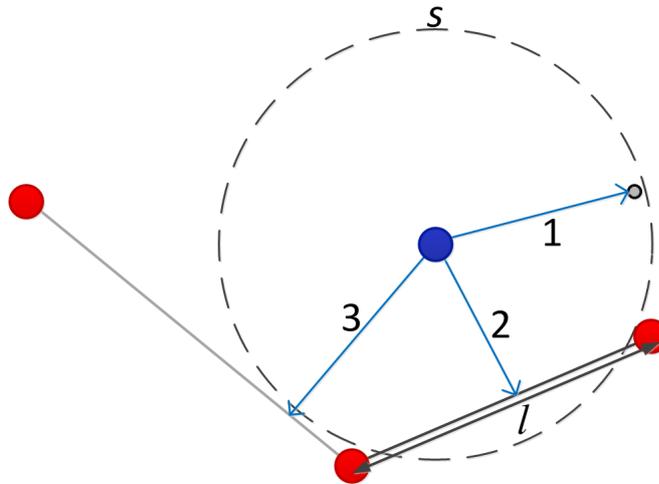


Figure 7.8: An example demonstrating the sight radius and rules which control the defender's behaviour. The three targets for the defenders are marked by the blue arrows and are numbered in order of their priority.

The control of defenders is rule based and these rules are summarised in Figure 7.8. Each defender has a sight radius s . The first priority is that if the ball is within s of the defender then it will move towards it as seen in Figure 7.8 marked '1'. If this is not the case then it makes a list of all of the potential passes and shots within s , of these it moves towards the shortest edge so long as that edge has a length of less than l as seen in Figure 7.8 marked '2'. If there are no edges within s with a length of less than l then the defender will move towards the nearest edge as seen in Figure 7.8 marked '3'. We found these rules elicit a good chasing and shutting down of opportunities behaviour by the defenders which keeps the game dynamic while still leaving the emphasis of control of the motion on the attackers and our proposed controller.

7.3.4 Football System Experiments

To demonstrate the system in action we have provided some example runs of our system in Figure 7.9, Figure 7.10 and Figure 7.11. In Figure 7.9 there are three attackers against three defenders and we have included six steps from a complete game. Here it can be seen that the ball is passed first to the leftmost attacker, then to the rightmost, providing it with a clear path to the goal avoiding all defenders. It can also be seen that over the course of the motion the formation is stretched along the y-axis and flattened

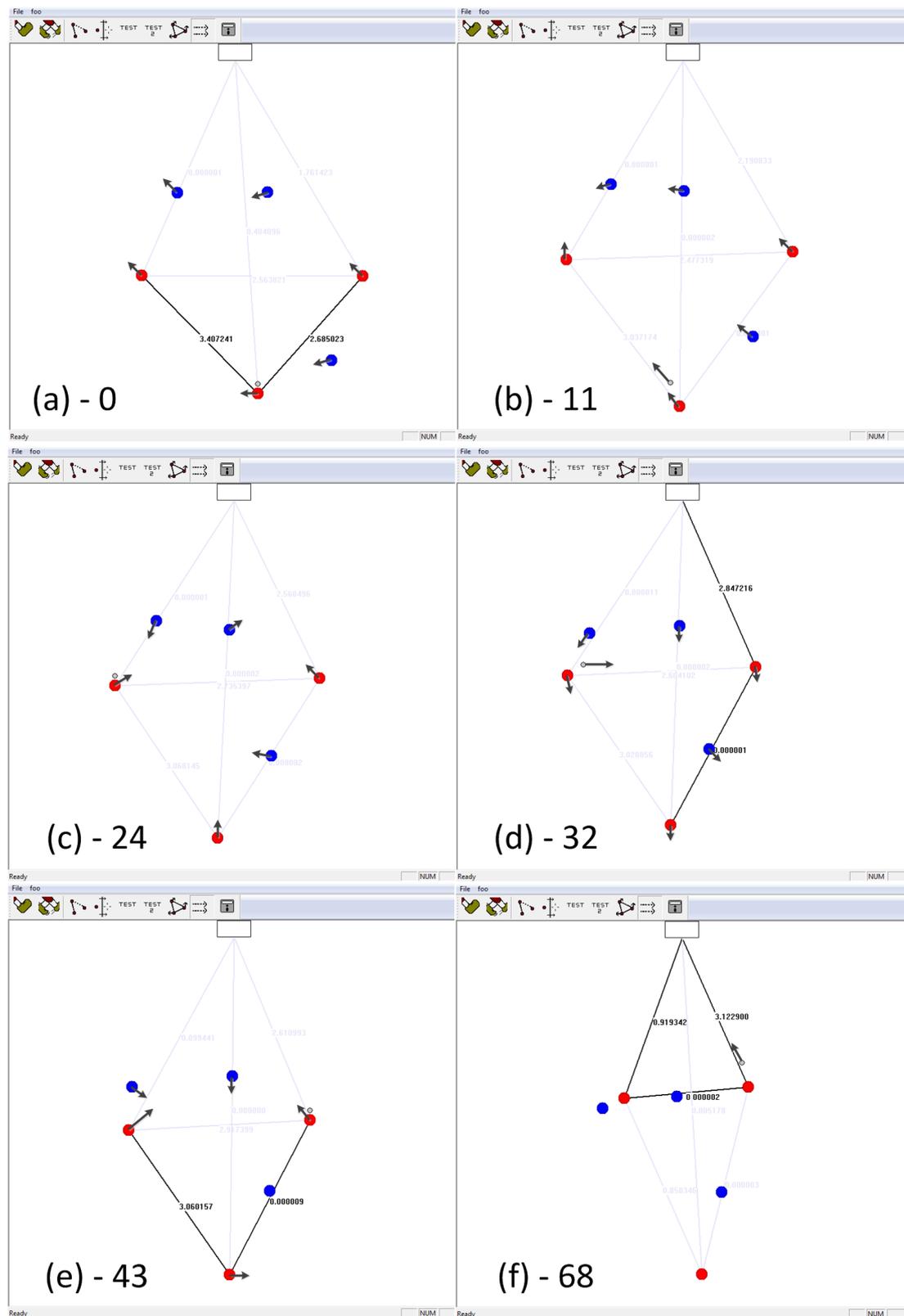


Figure 7.9: An example football game played out using our football controller. The time step numbers are given for each image and the movement of the players and ball in each scene are shown with arrows. The passes can be seen happening in images (b) and (d) and the final successful shot can be seen being initialised in image (f).

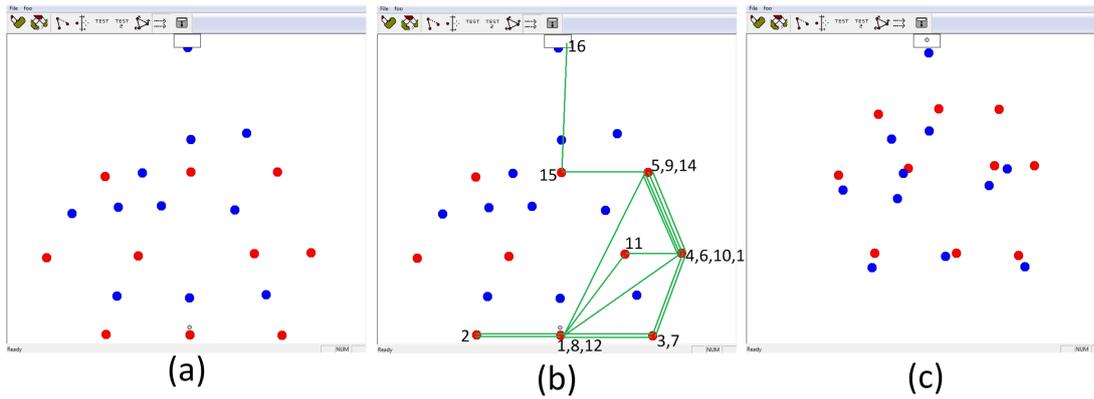


Figure 7.10: An example football scene with the full 10 attackers against 11 defenders. Image (a) shows the start state. Image (b) shows a play by play of the game, with the passes marked out by green edges and the numbering indicating the order in which players had possession of the ball after these passes (so the bottom centre player was the 1st, 8th and 12th player to hold the ball). Image (c) shows the final state of the game after 129 iterations.

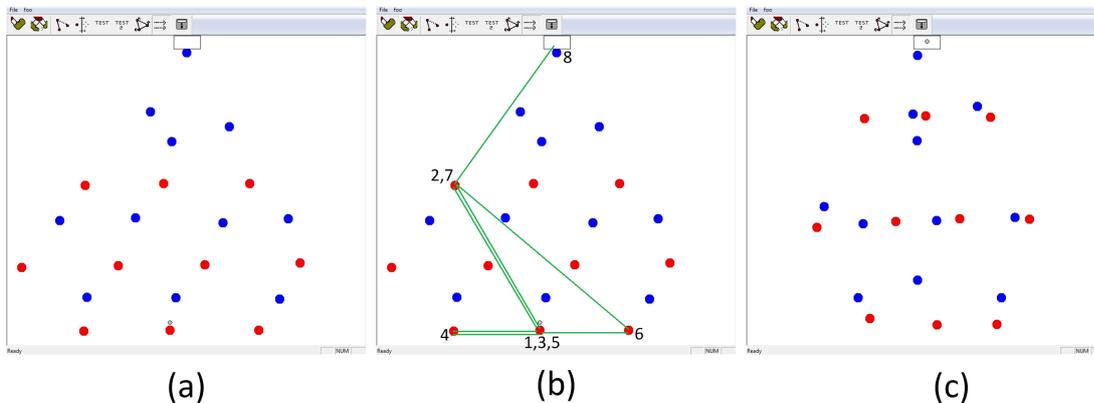


Figure 7.11: An example football scene with the full 10 attackers against 11 defenders. Image (a) shows the start state. Image (b) shows a play by play of the game, with the passes marked out by green edges and the numbering indicating the order in which players had possession of the ball after these passes. Image (c) shows the final state of the game after 81 iterations.

on the x-axis in order to better facilitate this process at each stage. Figure 7.10 and Figure 7.11 both show larger examples with a full compliment of attackers and defenders. It is worth pointing out that in both of these scenes there is a defender in goal, this player does not have an influence on the value of the shots which pass by it. This is because otherwise it effects every pass in the game and has a disproportionate influence on the behaviour of the entire formation of players. We believe this is justified as in football the focus is on setting up a shot on goal, with the position of the goalie when that shot happens being a lesser concern than getting the ball past the defenders. In Figure 7.10 it can be seen that the ball is passed up the less defended right side and then manoeuvred past the two defenders near the goal, the final formation gives an idea of how the players moved in order to bypass these defenders. In Figure 7.11 again a gap in the defence is exploited, but this time it is through the centre of the field. We believe these scenes show not only the ability of our system to cope with the most difficult problems which team games can offer, but also that they produce dynamic and reactive motion while doing so.

The method is fast, with an average running time ¹ per iteration of 27ms in the larger examples (as shown in Figure 7.10 and Figure 7.11, both with 11 attackers). However, there is significant variation to this value. This variation is caused by the iterations where the ball is in motion the attackers all just follow the current gradient, as these are almost instantaneous. The slowest iterations are those which use the attacker control, as described in Section 7.3.2, to find the new formation. If the passing is disabled then every iteration updates the attacker formation using the attacker control and the average iteration time is 58ms, with a standard deviation of 7ms. As these times are for the case containing 11 attackers (a full football team) they represent the worst case for the system. With fewer attackers the system is much faster, for instance in the example in Figure 7.9 each iteration takes only 8ms (with a standard deviation of 2ms).

7.3.5 Football System Conclusion

Within the field of computer graphics sports are a rarely tackled problem. In Shum et al. (2008) they give brief examples of how their multi-character control can apply to American football, but it is in terms of the interactions between pairs of players. Equally in Takahashi et al. (2009) they give some examples where they use their

¹All experiments were run on one core of an Intel core Duo E8400 with 4GB of ram

method to move characters into particular football formations, but these are keyframed formations which are provided for the system. There are a number of games, such as *FIFA* and *Madden*, which focus on sports. However, despite this and the large amount of research published in producing AI for games it is rare that sporting games are considered. As discussed in McGee and Abraham (2010), what approaches there are normally use reinforcement learning. This can be to many different purposes for instance in Mozgovoy and Umarov (2011) they focus on learning believable behaviour for football games from data of real players playing (and so mimicking behaviours like hesitance and uncertainty). In Laviers et al. (2009) and Laviers and Sukthankar (2011) they focus on American football and learning opponent behaviours such that the best and most appropriate team behaviour can be chosen. This case highlights one of the interesting problems, that for reinforcement learning, particularly for teams, the state evaluation needs some method of evaluating the reward. For American football this is a simple matter, as the success or failure of a play can be measured in how many yards were gained or lost. However, most sports are more continuous, so for regular football the value of a series of passes is not so clearly defined unless it results in a goal. Our method provides an immediate and simple way to provide such evaluations.

The main other area of comparison with our system is in robotics, specifically in Robocup. Introduced in Kitano et al. (1997) this is a competition to provide control for various classes of robots in an adapted form of football. Here much of the work is done on sub-problems of the football control problem such as keepaway, where one set of players try to keep possession of the ball against another set, as seen in Stone et al. (2005) and Jung and Polani (2012), or on half field offence, where players attack over half a football pitch, as in Kalyanakrishnan et al. (2006). There are two difficulties involved with comparing our system to these approaches. The first is that they deal with a different problem, the robotic nature of the Robocup competition means that they must cope with sensor noise, incomplete information, noisy actions and limited communications between agents. The second is that they are primarily hybrid methods using a number of approaches and heuristics and little global control, this is due to the high dimensionality of the problem and the high state space. However, here the reward structure is equally a problem and ongoing area of research, as shown in Devlin et al. (2011) and so again we believe that our graph based metric for evaluation could be of some assistance in the training of agents. As a result our method is complimentary to theirs and could act as an evaluation metric for global coordination, with the formation which we provide being used as a guide within which each robot could perform its

local control. For future work we would be interested in adapting our system for use in reinforcement learning for both gaming and robocup.

Within this football domain there are occasional cases which cause problems for our model. In particular if all of the defenders are off to one side of the attacking formation, then the formation will move off in the other direction even to the point of crowding up against the edge of the allowable space. This is obviously undesirable, though not a serious problem as it is reasonable to expect the defenders to be distributed evenly throughout the attacking formation. We have shown the effectiveness of our method on the particular football problem which we presented, clearly it also generalises. In the first case it could be used in a similar type of controller to decide the movement of a defending team. Also with very few changes it could also be used to control a basketball or hockey team and in fact the graph representation applies to any team game which involves the passing of some ball or puck.

Chapter 8

Conclusion

8.1 Contributions

We present a global controller which allows easy creation of crowd scenes where the crowd movement is both dynamic across the entire course of the motion and where the crowd is specifically tuned to fill the scene to its capacity. By filling the scene up to, but not beyond, its capacity this also allows simple avoidance of congestion at bottlenecks with no further effort on the part of the animator. Where by capacity we mean the available capacity across the routes defined by the start and end points, that is the maximal number of agents who can be sent through the scene without causing congestion. We further adapted this method to provide intuitive control over the entire crowd. Unlike most previous approaches, these controls can be seen to not just alter the behaviour of individual agents, but rather that of the crowd as a whole. Finally, we demonstrated the potential that this system has when adapted to other problems. Firstly, to the design of buildings with evacuation in mind, a problem to which it is already well tailored and could be applied towards with only a few small changes. Secondly, to team sports, a wide ranging control problem, which we demonstrated that the ideas behind this system could be adapted to handle with some modifications.

8.2 The Bigger Picture

At this point we felt it was important to give some sense of where our proposed system fits in with the corpus of crowd simulation research. It does not in fact represent a replacement for most current systems, rather it is a compliment. The paths provided to agents by our system could be followed by the vast majority of local controllers.

Rather it is a step towards providing coordinating control for a scene over its entire duration particularly for cases where dynamic scenes are required. Through the global variables computed it also provides globally based controls with which to alter the crowd.

We believe that this represents a promising direction of new research for crowd simulation. Local realism is a desirable property, but of far more use is allowing animators to effectively create functional scenes which fit the specifications which they have and localised control offers very little potential in this area. Tools such as those provided here or those present in Patil et al. (2011) provide a much simpler route for animators to produce a good initial scene, which they can then develop and tweak into a final product. This does not mean that local controllers are not important, but rather that higher level control which can compliment them is also needed and has thus far been neglected.

8.3 Future Work

There are a number of potentially promising directions for future work. The first and biggest of these is that we believe there is high potential for the use of our system in games, particularly in real-time strategy games. This is the case because these involve a problem which is almost identical to the one we are solving here, that is, moving a group of agents from a current location or set of locations to a single target in the most efficient way possible, while avoiding local obstacles. Comparison with modern versions of such games demonstrates that our system solves these problems much more efficiently than the current solutions. However, in order to apply our system properly to such areas it would obviously need to be significantly faster as the demands of such programs dictate that it should be not just real-time, but ideally much faster (to allow for all the further calculations also happening in such games). We believe that it is possible to achieve this performance through a number of adaptations. Firstly, by treating the Reeb graph as a navigation mesh, with each region representing a section of the mesh, and using this for navigation of the agents. Secondly, by performing the computation at multiple levels, providing a very low level rough calculation to get agents moving as quickly as possible, and refining this movement with a more high level recalculation once it is available. Finally, simply by providing a more optimised code which is more directed to this particular problem.

An additional advantage of adapting the system to work in real time is that there

might be potential for it to adaptively react to obstacles as they are placed. It could also allow other desirable features such as the use of non-homogeneous agents. Further experimentation would be needed to fully explore the potential of such a system.

Finally as alluded to in Section 8.2, we believe there is great potential for big strides to be made in global controllers for large crowds and associated tools. We propose that looking for further descriptors of a similar nature to those presented in Section 5.5, which allow for global manipulation of the entire crowd's behaviour would be a fruitful exploit.

Bibliography

- Almeida, R., Reis, L., and Jorge, A. (2009). Analysis and forecast of team formation in the simulated robotic soccer domain. In *Progress in Artificial Intelligence*, volume 5816, pages 239–250. Springer Berlin Heidelberg.
- Arechavaleta, G., Laumond, J.-P., Hicheur, H., and Berthoz, A. (2006). Optimizing principles underlying the shape of trajectories in goal oriented locomotion for humans. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 131–136.
- Barnett, A., Myung, C., and Taku, K. (2013). Topology-based global crowd control. In *The 26th International Conference on Computer Animation and Social Agents*.
- Bayazit, O. B. (2004). Roadmap-based flocking for complex environments. In *Proc. 10th Pacific Conference on Computer Graphics and Applications (PG02)*, pages 104–113.
- Berg, M. d. and Kreveld, M. J. v. (1993). Trekking in the alps without freezing or getting tired. In *Proceedings of the First Annual European Symposium on Algorithms, ESA '93*, pages 121–132, London, UK, UK. Springer-Verlag.
- Biasotti, S. (2004). Reeb graph representation of surfaces with boundary. In *Shape Modeling Applications, 2004. Proceedings*, pages 371 – 374.
- Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1124 –1137.
- Bradley, P. S., Carling, C., Archer, D., Roberts, J., Dodds, A., Di Mascio, M., Paul, D., Gomez Diaz, A., Peart, D., and Krustup, P. (2011). The effect of playing formation on high-intensity running and technical profiles in english fa premier league soccer matches. *Journal of Sports Sciences*, 29(8):821–830.

- Christensen, K. and Sasaki, Y. (2008). Agent-based emergency evacuation simulation with individuals with disabilities in the population. *Journal of Artificial Societies and Social Simulation*, 11(3):9.
- Clements, R. R. and Hughes, R. L. (2004). Mathematical modelling of a mediaeval battle: the battle of agincourt, 1415. *Math. Comput. Simul.*, 64(2):259–269.
- Connolly, C. I., Burns, J. B., and Weiss, R. (1990). Path planning using laplace’s equation. In *In Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 2102–2106.
- Davis, T. A. (2006). *Direct methods for sparse linear systems*. Society for Industrial and Applied Mathematics.
- Devlin, S., Grześ, M., and Kudenko, D. (2011). Multi-agent, reward shaping for robocup keepaway. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1227–1228. International Foundation for Autonomous Agents and Multiagent Systems.
- Dong, S., Kircher, S., and Garland, M. (2005). Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Comput. Aided Geom. Des.*, 22(5):392–423.
- Doraiswamy, H. and Natarajan, V. (2009). Efficient algorithms for computing reeb graphs. *Comput. Geom. Theory Appl.*, 42(6-7):606–616.
- Fiorini, P. and Shillert, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17:760–772.
- Fortune, S. (1987). A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1-4):153–174.
- Geraerts, R. (2010). Planning short paths with clearance using explicit corridors. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1997–2004. IEEE.
- Golas, A., Narain, R., and Lin, M. (2013). Hybrid long-range collision avoidance for crowd simulation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D ’13*, pages 29–36, New York, NY, USA. ACM.
- Guy, S. J., Chhugani, J., Curtis, S., Dubey, P., Lin, M., and Manocha, D. (2010). PLEdrians: a least-effort approach to crowd simulation. In *Proceedings of the*

- 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '10, pages 119–128, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.
- Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M. C., Manocha, D., and Dubey, P. (2009). Clearpath: Highly parallel collision avoidance for multi-agent simulation. In *ACM SIGGRAPH/EUROGRAPHICS SYMPOSIUM ON COMPUTER ANIMATION*, pages 177–187. ACM.
- Guy, S. J., Kim, S., Lin, M. C., and Manocha, D. (2011). Simulating heterogeneous crowd behaviors using personality trait theory. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '11*, pages 43–52, New York, NY, USA. ACM.
- Gwynne, S., Galea, E., Owen, M., Lawrence, P., and Filippidis, L. (2005). A systematic comparison of building exodus predictions with experimental data from the stapelfeldt trials and the milburn house evacuation. *Applied mathematical modelling*, 29(9):818–851.
- Helbing, D., Buzna, L., and Werner, T. (2003). Self-organized pedestrian crowd dynamics and design solutions. In *Traffic Forum 12*.
- Helbing, D., Farkas, I., and Vicsek, T. (2000). Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490.
- Helbing, D. and Molnár, P. (1995). Social force model for pedestrian dynamics. *Phys. Rev. E*, 51:4282–4286.
- Henderson, L. (1974). On the fluid mechanics of human crowd motion. *Transportation research*, 8(6):509–515.
- Hilaga, M., Shinagawa, Y., Kohmura, T., and Kunii, T. L. (2001). Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques, SIGGRAPH '01*, pages 203–212, New York, NY, USA. ACM.
- Hughes, R. L. (2002). A continuum theory for the flow of pedestrians. *Transportation Research Part B: Methodological*, 36(6):507 – 535.

- Hughes, R. L. (2003). The flow of human crowds. *Annual review of fluid mechanics*, 35(1):169–182.
- Jiang, H., Xu, W., Mao, T., Li, C., Xia, S., and Wang, Z. (2010). Continuum crowd simulation in complex environments. *Computers & Graphics*, 34(5):537–544.
- Ju, E., Choi, M. G., Park, M., Lee, J., Lee, K. H., and Takahashi, S. (2010). Morphable crowds. In *ACM SIGGRAPH Asia 2010 papers*, SIGGRAPH ASIA '10, pages 140:1–140:10, New York, NY, USA. ACM.
- Jund, T., Kraemer, P., and Cazier, D. (2012). A unified structure for crowd simulation. *Comput. Animat. Virtual Worlds*, 23(3-4):311–320.
- Jung, T. and Polani, D. (2012). Learning robocup-keepaway with kernels. *CoRR*, abs/1201.6626.
- Kalyanakrishnan, S., Liu, Y., and Stone, P. (2006). Half field offense in robocup soccer: a multiagent reinforcement learning case study. In *Proceedings of the RoboCup International Symposium 2006*. Springer Verlag.
- Kapadia, M., Singh, S., Hewlett, W., and Faloutsos, P. (2009). Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 215–223, New York, NY, USA. ACM.
- Kapadia, M., Wang, M., Singh, S., Reinman, G., and Faloutsos, P. (2011a). Scenario space: characterizing coverage, quality, and failure of steering algorithms. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, pages 53–62, New York, NY, USA. ACM.
- Kapadia, M., Wang, M., Singh, S., Reinman, G., and Faloutsos, P. (2011b). Scenario space: characterizing coverage, quality, and failure of steering algorithms. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 53–62. ACM.
- Karamouzas, I., Geraerts, R., and Overmars, M. (2009). Indicative routes for path planning and crowd simulation. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 113–120, New York, NY, USA. ACM.

- Karamouzas, I., Geraerts, R., and van der Stappen, A. F. (2013). Space-time group motion planning. In *Algorithmic Foundations of Robotics X*, pages 227–243. Springer.
- Ketchell, N., Cole, S., Webber, D., Marriott, C., Stephens, P., Brearley, I., Fraser, J., Doheny, J., and Smart, J. (1993). *The EGRESS code for human movement and behaviour in emergency evacuations*. University of Edinburgh, Artificial Intelligence Applications Institute.
- Kim, S., Guy, S. J., Manocha, D., and Lin, M. C. (2012). Interactive simulation of dynamic crowd behaviors using general adaptation syndrome theory. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 55–62, New York, NY, USA. ACM.
- Kisko, T. M. and Francis, R. L. (1985). Evacnet+: a computer program to determine optimal building evacuation plans. *Fire Safety Journal*, 9(2):211–220.
- Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1997). Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, pages 340–347. ACM.
- Klüpfel, H. and Meyer-König, T. (2003). Characteristics of the pedgo software for crowd movement and egress simulation. In *2nd International Conference in Pedestrian and Evacuation Dynamics (PED)*, pages 331–340.
- Kovar, L., Gleicher, M., and Pighin, F. (2002). Motion graphs. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 473–482, New York, NY, USA. ACM.
- Krogh, B. (1984). *A Generalized Potential Field Approach to Obstacle Avoidance Control*. Creative manufacturing engineering program. RI/SME.
- Kwon, T., Lee, K. H., Lee, J., and Takahashi, S. (2008). Group motion editing. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 80:1–80:8, New York, NY, USA. ACM.
- Lai, Y.-C., Chenney, S., and Fan, S. (2005). Group motion graphs. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 281–290, New York, NY, USA. ACM.

- Lamarche, F. and Donikian, S. (2004). Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 23:509–518.
- Laumond, J.-P., Arechavaleta, G., Truong, T.-V.-A., Hicheur, H., Pham, Q.-C., and Berthoz, A. (2011). The words of the human locomotion. In *Robotics Research*, pages 35–47. Springer.
- Laviers, K. and Sukthankar, G. (2011). A real-time opponent modeling system for rush football. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2476–2481. AAAI Press.
- Laviers, K., Sukthankar, G., Aha, D. W., Molineaux, M., Darken, C., et al. (2009). Improving offensive performance through opponent modeling. In *AIIDE*.
- Lee, K. H., Choi, M. G., Hong, Q., and Lee, J. (2007). Group behavior from video: a data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 109–118, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.
- Lee, S., Son, Y.-J., and Jin, J. (2010). An integrated human decision making model for evacuation scenarios under a bdi framework. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 20(4):23.
- Lerner, A., Chrysanthou, Y., and Lischinski, D. (2007). Crowds by example. *Computer Graphics Forum*, 26(3):655–664.
- Lerner, A., Fitusi, E., Chrysanthou, Y., and Cohen-Or, D. (2009). Fitting behaviors to pedestrian simulations. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 199–208, New York, NY, USA. ACM.
- Lo, S., Fang, Z., Lin, P., and Zhi, G. (2004). An evacuation model: the sgem package. *Fire Safety Journal*, 39(3):169–190.
- Maïm, J., Yersin, B., Thalmann, D., and Pettré, J. (2009). Yaq: an architecture for real-time navigation and rendering of varied crowds. *IEEE Comput. Graph. Appl.*, 29(4):44–53.

- McGee, K. and Abraham, A. T. (2010). Real-time team-mate ai in games: A definition, survey, & critique. In *proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 124–131. ACM.
- Morini, F., Yersin, B., Maym, J., and Thalmann, D. (2007). Real-time scalable motion planning for crowds. In *Proceedings of the 2007 International Conference on Cyberworlds, CW '07*, pages 144–151, Washington, DC, USA. IEEE Computer Society.
- Mozgovoy, M. and Umarov, I. (2011). Believable team behavior: Towards behavior capture ai for the game of soccer. In *8th International Conference on Complex Systems*, pages 1554–1564.
- Musse, S. R. and Thalmann, D. (1997). A model of human crowd behavior: Group inter-relationship and collision detection analysis. In *Proc. Workshop of Computer Animation and Simulation of Eurographics97*, pages 39–51.
- Musse, S. R. and Thalmann, D. (2001). Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, 7:152–164.
- Narain, R., Golas, A., Curtis, S., and Lin, M. C. (2009). Aggregate dynamics for dense crowd simulation. In *ACM SIGGRAPH Asia 2009 papers, SIGGRAPH Asia '09*, pages 122:1–122:8, New York, NY, USA. ACM.
- NFPA (2006). National fire protection association 101. *Life Safety Code 2006 edition*.
- Ni, X., Garland, M., and Hart, J. C. (2004). Fair morse functions for extracting the topological structure of a surface mesh. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 613–622. ACM.
- Overmars, M. H. (1992). A random approach to motion planning. *RUU-CS*, (92-32).
- Patil, S., van den Berg, J., Curtis, S., Lin, M. C., and Manocha, D. (2011). Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):244–254.
- Pelechano, N., Allbeck, J. M., and Badler, N. I. (2007). Controlling individual agents in high-density crowd simulation. In *Proc. of ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA)*.

- Pettré, J., Ciechomski, P. d. H., Maïm, J., Yersin, B., Laumond, J.-P., and Thalmann, D. (2006). Real-time navigating crowds: scalable simulation and rendering: Research articles. *Comput. Animat. Virtual Worlds*, 17(3-4):445–455.
- Pettré, J., Laumond, J.-P., and Thalmann, D. (2005). A navigation graph for real-time crowd animation on multilayered and uneven terrain. In *First International Workshop on Crowd Simulation*, volume 43, page 194. Citeseer.
- Pettré, J., Ondřej, J., Olivier, A.-H., Cretual, A., and Donikian, S. (2009). Experiment-based modeling, simulation and validation of interactions between virtual walkers. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 189–198, New York, NY, USA. ACM.
- Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a bdi-architecture. *KR*, 91:473–484.
- Reeb, G. (1946). Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique. *Comptes Rendus de L'Académie des Séances, Paris*, 222:847–849.
- Reynolds, C. W. (1987a). Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34.
- Reynolds, C. W. (1987b). Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34.
- Shao, W. and Terzopoulos, D. (2005). Autonomous pedestrians. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 19–28, New York, NY, USA. ACM.
- Shao, W. and Terzopoulos, D. (2006). Populating reconstructed archaeological sites with autonomous virtual humans. In *Proceedings of the 6th international conference on Intelligent Virtual Agents*, IVA'06, pages 420–433, Berlin, Heidelberg. Springer-Verlag.
- Shi, J., Ren, A., and Chen, C. (2009). Agent-based evacuation model of large public buildings under fire conditions. *Automation in Construction*, 18(3):338–347.
- Shinagawa, Y., Kunii, T., and Kergosien, Y. (1991). Surface coding based on morse theory. *Computer Graphics and Applications, IEEE*, 11(5):66–78.

- Shum, H. P. H., Komura, T., Shiraishi, M., and Yamazaki, S. (2008). Interaction patches for multi-character animation. In *ACM SIGGRAPH Asia 2008 papers*, SIGGRAPH Asia '08, pages 114:1–114:8, New York, NY, USA. ACM.
- Silveira, R., Dapper, F., Prestes, E., and Nedel, L. (2010a). Natural steering behaviors for virtual pedestrians. *Vis. Comput.*, 26(9):1183–1199.
- Silveira, R., Fischer, L., Ferreira, J. A. S., Prestes, E., and Nedel, L. (2010b). Path-planning for rts games based on potential fields. In *Proceedings of the Third international conference on Motion in games*, MIG'10, pages 410–421, Berlin, Heidelberg. Springer-Verlag.
- Singh, S., Kapadia, M., Hewlett, B., Reinman, G., and Faloutsos, P. (2011). A modular framework for adaptive agent-based steering. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 141–150 PAGE@9, New York, NY, USA. ACM.
- Snape, J., van den Berg, J., Guy, S., and Manocha, D. (2011). The hybrid reciprocal velocity obstacle. *Robotics, IEEE Transactions on*, 27(4):696–706.
- Snook, G. (2000). Simplified 3d movement and pathfinding using navigation meshes. *Game Programming Gems*, 1:288–304.
- Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188.
- Sung, M., Gleicher, M., and Chenney, S. (2004). Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23(3):519–528.
- Takahashi, S., Yoshida, K., Kwon, T., Lee, K. H., Lee, J., and Shin, S. Y. (2009). Spectral-based group formation control. In *Computer Graphics Forum*, volume 28, pages 639–648. Wiley Online Library.
- Torchelsen, R. P., Scheidegger, L. F., Oliveira, G. N., Bastos, R., and Comba, J. a. L. D. (2010). Real-time multi-agent path planning on arbitrary surfaces. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, I3D '10, pages 47–54, New York, NY, USA. ACM.
- Treuille, A., Cooper, S., and Popovi, Z. (2006). Continuum crowds. *ACM Trans. Graph*, 25:1160–1168.

- Tu, X. and Terzopoulos, D. (1994). Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 43–50.
- van den Akker, M., Geraerts, R., Hoogeveen, H., and Prins, C. (2010). Path planning for groups using column generation. In *Proceedings of the Third international conference on Motion in games*, MIG'10, pages 94–105, Berlin, Heidelberg. Springer-Verlag.
- van den Berg, J., Lin, M. C., and Manocha, D. (2008a). Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, pages 1928–1935. IEEE.
- van den Berg, J., Patil, S., Sewall, J., Manocha, D., and Lin, M. C. (2008b). Interactive navigation of multiple agents in crowded environments. In *SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES*, pages 139–147. ACM.
- van den Berg, J., Patil, S., Sewall, J., Manocha, D., and Lin, M. C. (2008c). Interactive navigation of multiple agents in crowded environments. In *SYMPOSIUM ON INTERACTIVE 3D GRAPHICS AND GAMES*, pages 139–147. ACM.
- van Kreveld, M., van Oostrum, R., Bajaj, C., Pascucci, V., and Schikore, D. (1997). Contour trees and small seed sets for isosurface traversal. In *Proceedings of the thirteenth annual symposium on Computational geometry*, SCG '97, pages 212–220, New York, NY, USA. ACM.
- van Toll, W. G., Cook, A. F., and Geraerts, R. (2012a). A navigation mesh for dynamic environments. *Comput. Animat. Virtual Worlds*, 23(6):535–546.
- van Toll, W. G., Cook, IV, A. F., and Geraerts, R. (2012b). Real-time density-based crowd simulation. *Comput. Animat. Virtual Worlds*, 23(1):59–69.
- Xiao, Y., Siebert, P., and Werghi, N. (2003). A discrete reeb graph approach for the segmentation of human body scans. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on*, pages 378 – 385.
- Yeh, H., Curtis, S., Patil, S., van den Berg, J., Manocha, D., and Lin, M. (2008). Composite agents. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 39–47, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

- Yersin, B., Maïm, J., Pettré, J., and Thalmann, D. (2009). Crowd patches: populating large-scale virtual environments for real-time applications. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games, I3D '09*, pages 207–214, New York, NY, USA. ACM.
- Yu, Q. and Terzopoulos, D. (2007). A decision network framework for the behavioral animation of virtual humans. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, SCA '07*, pages 119–128, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.
- Zhou, S., Chen, D., Cai, W., Luo, L., Low, M. Y. H., Tian, F., Tay, V. S.-H., Ong, D. W. S., and Hamilton, B. D. (2010). Crowd modeling and simulation technologies. *ACM Trans. Model. Comput. Simul.*, 20(4):20:1–20:35.