Evolving Controllable Emergent Crowd Behaviours with Neuro-Evolution

A dissertation submitted in satisfaction of the requirements for the degree Master of Science in Computer Science

by

Sunrise Wang

Supervised by:

James Gain and Geoff Nitschke

June 2015

I know the meaning of plagiarism and declare that all of the work in this document, save for that which is properly acknowledged, is my own.

Acknowledgements

Firstly, I would like to thank my supervisors James Gain and Geoff Nitschke for their support, guidance, and feedback provided throughout the past two and a half years.

I would also like to thank all the people in the Procedural Interest Group for both sharing their work, and providing feedback. I must also thank my lab mates for the joining me in the procrastination circles, you have helped me retain my sanity.

I also want to thank my parents for their support and financial backing provided throughout the course of my studies.

Finally, I want to thank the National Research Foundation for funding this thesis through the Technology and Human Resources for Industry Programme in the form of THRIP grant TP2011071600004.

Abstract

Crowd simulations have become increasingly popular in films over the past decade, appearing in large crowd shots of many big name block-buster films. An important requirement for crowd simulations in films is that they should be directable both at a high and low level, and be believable. As agent-based techniques allow for low-level directability and more believable crowds, they are typically used in this field. However, due to the bottom-up nature of these techniques, achieving high level directability requires the modification of agent-level parameters until the desired crowd behaviour emerges.

As manually adjusting parameters is a time consuming and tedious process, this thesis investigates a method for automating this, using Neuro-Evolution (NE). This is achieved by using Artificial Neural Networks as the agent controllers within an animated scene, and evolving these with an Evolutionary Algorithm so that the agents behave as desired. To this end, this thesis proposes, implements, and evaluates a system that allows for the low-level control of crowds using NE. Overall, this approach shows very promising results, with the time taken to achieve the desired crowd behaviours being either on par or faster than previous methods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Crowds are commonplace in real life and can be observed in a variety of forms, such as bird flocks, pedestrians, concert audiences, and traffic. Due to their ubiquity, it is important for virtual environments to simulate them in order to appear realistic and believable. Traditional animation approaches struggle in this context because of the large amount of work required for artists to animate not only the behaviours of each individual agent, but also their interactions with other agents. Crowd Simulations are a set of computational techniques that deal with this problem by automating many of the animations of the agents within the environment.

Since the introduction of Reynolds' seminal work on bird flocking [Rey87], there has been considerable research performed on Crowd Simulations. This research can be roughly divided into two groups: *Macroscopic* [PPD07, NGCL09] and *Microscopic* [THH00, HBD02] approaches.

Macroscopic approaches aim to simulate crowds in a top-down manner, by performing the simulation at a group level. These approaches typically use concepts such as flow, fluid dynamics, and density fields in order to simulate the behaviour of crowds. *Macroscopic* techniques are advantageous in that they can typically simulate larger groups of crowds in real time compared to their *Microscopic* counterparts, as well as provide an easy way to direct the aggregate motion of the crowds.

Microscopic approaches simulate crowds in a bottom-up manner, by simulating the agents on an individual basis with the aggregate crowd behaviour being emergent. Advantages to these techniques are that agents are capable of interacting within the crowd and that individual agents can be controlled and directed. This leads to the overall crowd being more believable.

Crowds are used in a variety of applications such as digital entertainment [MUAT05, HDK+06], psychology [MPT92, JPVdS01, TSM99], architectural design [SOHTG99, PT01, TP02], and military training [Woo00, PMGW04]. Given this wide variety of applications, it is natural that each has differing requirements. For example when simulating evacuations in architecture, the individual agents tend to have rather simple behaviours with the overall motion of the crowd being the main focus, whereas in Collaborative Virtual Environments, the crowd must be simulated in real-time as it needs to interact with participants [THL+04].

In the film industry, the crowds (high-level) and individual agents (low-level) should be controllable, and their large scale behaviour believable [THL+04]. In order to cater

Figure 1.1: *Images from left to right, top to bottom: "The Lord of the Ring - Return of the King", "Happy Feet", "I, Robot", and "The Dark Knight" (Source: [CSM]).*

for the low-level control and believability requirement, systems employing Microscopic techniques are typically used. The industry standard, *Massive*[1], is one such system, and has been used to animate crowd shots in a host of films, including "The Lord of the Rings", "Happy Feet", "I, Robot", and "The Dark Knight" (figure 1.1). *Massive* simulates these crowds using Fuzzy Systems [BT97] for the the individual agent brains, which determine the agents' various behaviours and animations.

One problem that microscopic systems, such as *Massive*, suffer from is accomplishing the third goal of films, *high-level control*. This is expected, as high-level control is an opposing requirement to believable crowds [THL+04]. Traditionally, this problem is solved by artists tuning the low level parameters of the *Fuzzy Systems* until the desired behaviours emerge - a tedious and time consuming task.

One method for dealing with this is to use optimization algorithms to find the ideal parameters so that the high-level crowd behaviour is as desired. This is particularly relevant in films due to three characteristics: shots are not rendered in real-time thus there is no real-time requirement; there is no shot to shot coherence; shots are short [THL+04].

An example of such a method is Jacka's work [Jac09], where he used Particle Swarm Optimization [EK95] to optimize the membership functions of the Fuzzy Systems representing the agents' brains.

This thesis is focused on exploring another paradigm, namely Neuro-Evolution (NE). NE is the use of Evolutionary Algorithms [D+91] to evolve Artificial Neural Networks (ANNs) [Yeg09]. This method can be applied to high-level crowd control by using ANNs as the brains of the agents within a Crowd Simulation, and then evolving these ANNs so that the crowd and agent behaviours reflect the desired user specifications, which are represented as a set of objectives in the fitness function of the NE algorithm.

[1] http://www.massivesoftware.com/

This thesis' intent is to investigate how well NE performs when applied to high-level crowd simulation control. Since there are many NE algorithms, each with a wide range of parameters that have problem dependent task performance, we also aim to investigate the ideal algorithms and parameters for this specific problem.

The main contribution of this thesis is a novel NE method of providing high-level control to Crowd Simulations, as well as a new application area in which to study NE. Such an approach has the advantage over Fuzzy System membership function optimization in that artists are not required to design the brains but rather merely provide objectives for the crowd simulation. This is desirable for two main reasons: it frees up more of the artists time, allowing them to be more productive; and that poorly designed brains may lead to difficulties in finding the desired behaviours. Additionally, compared to other control techniques, it allows for a much wider variety of agent types and objectives, making it a more general solution.

Overall, this thesis shows promising results when using NE to control aggregate emergent crowd behaviours, with the evolution times being on par or better than the times obtained in Jacka's work [Jac09]. We identified CMA-ES to be the best performing algorithm for high-level control of crowd simulations.

1.1 Research Goals

Our research goals for this project are as follows:

- Determine whether or not it is possible to control the emergent behaviours of Crowd Simulations using NE for a range of tasks and environments that require crowd behaviour.
- Determine if the evolution times are feasible so that this method can be used in the context of controlling crowd animations in films.
- Find an appropriate NE algorithm for controlling Crowd Simulations.

For the first research goal, we are interested in whether or not it is possible to evolve controllable behaviours for Crowd simulations using NE. This is important as NE algorithms often become trapped in poor local optima or prematurely converge, which may result in behaviours that do not reflect those specified by the user.

For the second research goal, we are interested in whether the evolution times when using NE to control crowds are feasible for possible use in films. This is important as, although the crowds are not rendered in real-time, the evolution process should not bottleneck the animation pipeline. In order to determine if the times are feasible, we will compare our evolution times to the expert user parameter tuning times obtained by Jacka [Jac09], who implemented a system very similar to *Massive*.

For the third research goal, we have decided to compare the task performance of four well established NE algorithms that have been successfully applied to a range of other task domains [SM02, GM03b, SBM05], together with their respective parameters and operators when evolving controllable high-level crowd behaviours. These algorithms are Neuro-Evolution of Augmenting Topologies [SM02] (section 4.1.4), Enforced Sub-Populations

[GM03b] (section 4.1.2), Conventional Neuro-Evolution [Wie91] (section 4.1.1), and Covariance Matrix Adaptation Evolutionary Strategy [HK04] (section 4.1.3). This is important as the task performance of NE algorithms and parameters often are problem dependent. These algorithms were selected as they represent the various categories of NE algorithms discussed in section 3.2. In order to evaluate how these algorithms perform, we will primarily look at how well and how quickly they are able to evolve the desired behaviours defined by the NE algorithm's fitness function.

1.2 Contributions

The contributions of this thesis are as follows:

- A novel NE method for controlling high-level behaviours in crowd-simulations, together with an investigation of the performance of NE in this problem area.
- An investigation of the task performance of NE crossover, selection, and mutation operators in the context of controlling crowd behaviours.
- A comparison of the task performance of four well established NE algorithms when controlling crowd behaviours.
- Insight into how ANN controller compositions affect the evolutionary process and the final evolved agent behaviours.

1.3 Scope

This thesis is primarily focused on exploring various NE mechanisms in the context of controlling crowd simulations. Due to the sheer variety of these mechanisms, several avenues of research have been excluded in order to limit scope. These include:

- The design and implementation of a user interface that allows users to easily use the system.
- An in-depth investigation of different user-provided crowd objectives, which may allow for more complex high-level control (for example crowd formations and inter-agent interactions), or faster evolution.
- Non-behavioural aspects of crowd simulation, such as optimizing rendering and animations.

Due to these exclusions, this work serves mainly as a preliminary study of the viability of using NE for controlling crowds. Further work will be required in the areas listed above before it can be used in a production environment.

1.4 Structure

This thesis is structured as follows. Chapters 2 and 3 provide background work and fundamentals required for the understanding of our research. These chapters explore previous work in the fields of crowd simulations and NE. Chapters 4 and 5 provide an overview on the design and implementation of our system, with in-depth elaborations on the various algorithms used. Chapter 6 details the design of the various simulations and agents used. Chapter 7 covers details of how we plan to conduct our experiments, as well as providing parameter tuning results and motivations. Chapter 8 provides insight with regards to the results obtained in the experiments. Chapter 9 concludes and gives future directions for our work.

Chapter 2

Crowd Simulations

Crowd Simulations are a set of techniques that simulate the collective behaviour of agents within virtual environments. In the last few decades, these techniques have been applied in a wide variety of fields, including architecture, computer graphics, physics, robotics, safety science, training, and sociology [Tha07].

Despite the broad range of applications, there has been a lack of cross pollination, resulting in most approaches being context specific, with different fields focusing on different aspects of crowd behaviour [Tha07]. Despite the differences in these approaches, their behavioural models can be divided roughly into two categories: *Macroscopic* approaches, which aim to simulate the crowd in a top-down manner, with the crowd being simulated at a group level, and *Microscopic* approaches, which simulate crowds bottom-up with the crowd being simulated at an agent-level. The differences in these approaches result in aggregate crowd behaviour being artist defined for Macroscopic approaches, and emergent for Microscopic ones.

This chapter provides an overview of the various techniques dealing with the behavioural modelling of crowds, examples of applications for crowd simulations, an overview of other important aspects of crowd simulations, such as rendering and authoring, and a discussion of crowd simulations in the context of films, our primary field of interest.

2.1 Crowd Simulation Applications

Crowd simulations have been used in a vast variety of fields, each having differing requirements. One field that uses crowd simulations extensively is sociology, where it is often used to study collective behaviour of pedestrians, such as arcs, rings, and clustering [MPT92, JPVdS01, TSM99]. Thus, it is important for crowd simulations in this field to model social structures, such as grouping and queuing. Microscopic approaches are preferred in this context as they better reflect the decision making process of real-life humans, and allow for interesting and sometimes unexpected behaviours to emerge from the crowd.

A large amount of crowd simulation work has also been performed in the field of architecture, where crowd simulations mainly aim to simulate pedestrians dynamics in order to accomplish tasks such as planning town centres and assessing the placement of shops [SOHTG99]. This is achieved by using the crowd simulations to predict and study the paths [PT01] and behaviours [TP02] of pedestrians in the relevant environments.

Figure 2.1: *The Zheng Zhi Adidas advertisement for the 2008 Olympic games (Source: [zhe]).*

Microscopic pedestrian simulations tend to perform well in this field as, once again, much of the research aims to study the emergent behaviours and properties of crowds.

Another application for crowd simulations in architecture, as well as safety sciences, is evacuation simulation [Tha07]. The main aim here is to model the movement and behaviours of large groups of people in confined spaces, often in the presence of some hazard. This enables architects and designers to more easily find flaws in their designs [OM93]. These simulations help the designer answer questions such as: *can the area be evacuated within a specified time?*; and *where are the flow bottlenecks?* [Tha07]. Thus, modelling the motion of agents is the most important aspect to these simulations. Both Microscopic and Macroscopic methods work well in this context, with the preferred technique being scenario dependent. Microscopic simulations, due to simulating individual agents, allow for more complex behaviours, such as shelter seeking or hiding, whereas Macroscopic simulations allow for large numbers of agents, due to reduced computational cost of simulating at a group level.

Virtual environments are an application area of crowd simulations that has been gaining traction recently, with examples ranging from battle scenes in *The Lord of the Rings* (figure 2.2) to crowds of spectators in the Adidas *Zheng Zhi* advertisement (figure 2.1). Crowd simulations are used in these virtual environments to enhance realism and immersiveness. The requirements in this field differ from other areas such as evacuations, sociology, and training in that it is important to have both believable behavioural models, as well as realistic animations and visualizations. Once again, the techniques used here are dependent on context. Microscopic techniques work well if the application does not require real-time response, only needs to simulate small crowds, or requires high qual-

Figure 2.2: *Battle of the Hornbug scene in The Lord of The Rings: The Two Towers.*

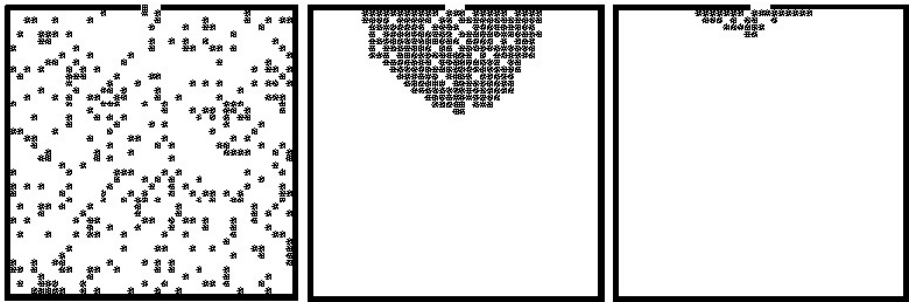

Figure 2.3: *An example of a Cellular Automata model used for evacuating agents from a room (Source: [KS02]).*

ity animations. Macroscopic simulations, on the other hand, are useful for applications where interactive simulation of large crowds is necessary.

2.2 Microscopic Crowd Simulations

An intuitive way of simulating crowds would be to define the behaviours of individual agents, and allow the aggregate behaviour of the crowd to emerge. This class of crowd simulation techniques has been termed Microscopic [THH00, HBD02], Bottom-up [Tha07, OPOD10], and Agent-based [YCP$^+$08, QH10]. These techniques can be divided up into three broad categories: Cellular automata systems, Rule-based systems, and Embodied agent systems.

2.2.1 Crowds with Cellular Automata

The use of Cellular Automata to model pedestrians has proved popular. In these techniques, the environment is modelled as a grid of cells (figure 2.3), with pedestrians and obstacles being able to occupy these cells. The pedestrians are then moved between cells based on a variety of factors, such as density, speed, and flow. One such system is *Legion*,

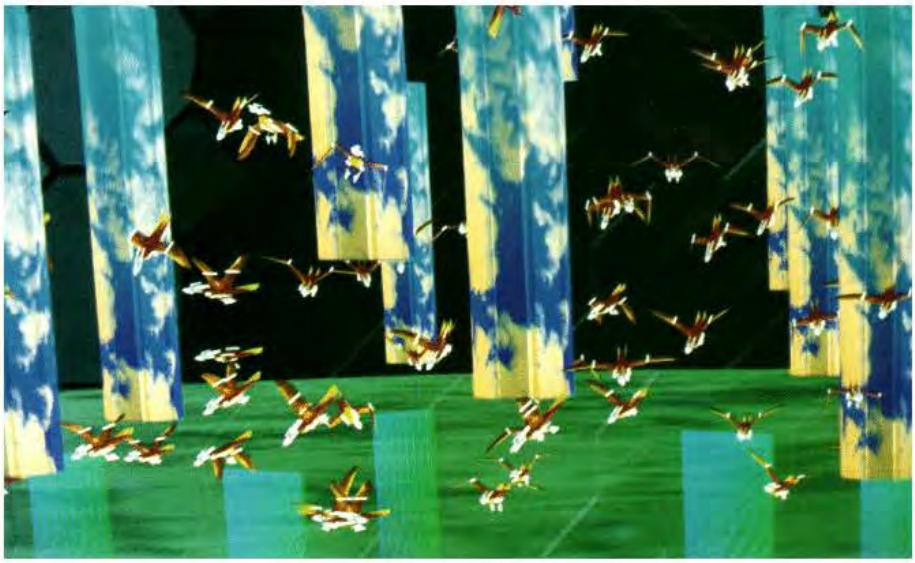

Figure 2.4: *Birds flocks in* ***Stanley and Stella in: Breaking the Ice*** *simulated using Reynolds' boids algorithm (Source: [Rey87]).*

a pedestrian simulation system inspired by the work of Still [Sti00], and has been used for a variety of tasks, such as the planning of stadiums during the Olympic games. In *Legion*, an agent transitions between the cells by taking into account its position as well its local environment.

Cellular Automata crowd simulation systems have been used to study several scenarios, such as evacuating ships [KMKWS01], effects of crowds on a person's mood [JPVdS01], and self-organization and phase transitions amongst pedestrians [FI99]. Despite the success of these systems in studying pedestrian behaviour, they are, unfortunately, limited to applications that do not require highly realistic visualizations, such as film, because their inherently discrete nature results in limitations as to how the agents can move around the environment, often resulting in less believable behaviours.

2.2.2 Rule-based systems

The short film *Stanley and Stella in: Breaking the Ice*, shown at the Electronic Theater at SIGGRAPH '87, showcased the revolutionary idea that complex group behaviors can be achieved by providing simple local rules to the agents within the group, as opposed to using some set of enforced global rules (figure 2.4). Accompanying this short film was a technical paper detailing the technique used to simulate the flocking behavior. Reynolds' paper on boids [Rey87] is now widely considered the first crowd simulation technique.

In boids, each agent is provided with a local perception of the environment, as well as three simple local rules: cohesion, separation, and alignment. Alignment requires that an agent should try to match the velocity of its neighbouring agents, separation means that the agents should attempt to steer away from their neighbours, and cohesion tells the agents to move towards the centroid of the group. Through these three simple local rules, Reynolds showed that it was possible for complex aggregate behaviour to emerge within the flock.

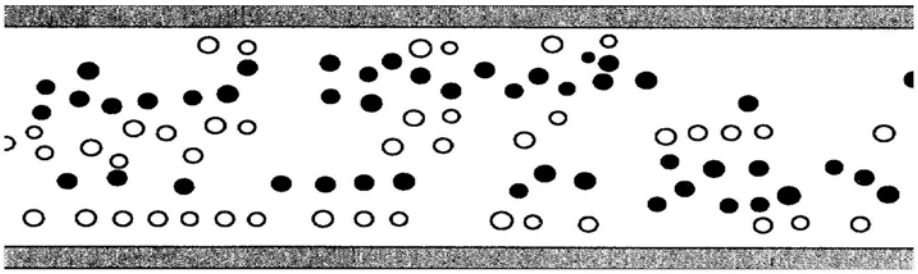

Figure 2.5: *Lane segregation of agents in Helbing and Molnar's Social Forces model (Source: [HM95]).*

Reynolds has since extended boids to include additional behaviours, such as goal seeking, obstacle avoidance, path following, and fleeing [Rey99], as well as incorporating finite state machines into the agent controller model in order to allow for interactions between groups and users.

Another well known and often used technique is Helbing and Molnar's [HM95] particle-based *Social Force Model*. In their technique, each particle represents an agent within the crowd, and is provided a mass and a desired velocity. These desired velocities are changed throughout the simulations through the use of two external forces: force to move agents away from walls; and force to move agents away from other agents. Helbing and Molnar found that with this model, certain emergent behaviours, such as lane segregation (figure 2.5), formed.

A problem with the social forces model is that all the agents have the same behavioral rules. Braun et al. deal with this by extending Helbing's model to allow for the modelling of different behaviours for individuals and groups. They achieved this by adding terms such as altruism and dependency levels to the model [BMdOB03], as well as providing agents with the ability to perceive nearby emergencies and alarms [BBM05].

A recent direction for crowd simulation research is modelling the high-level psychological state of agents. A well known approach in this area is the High-Density Autonomous Crowds (HiDAC) system by Pelechano et al. [PAB07]. HiDAC utilizes a 2-tiered architecture for simulating agents, with the upper-tier defining high-level functions, such as navigation, communication, decision making, and learning, and the lower-tier defining low-level motion, such as reactive behaviours and perception (figure 2.6).

In order to simulate the agents, the system implements a parametrized version of the Social Forces model [HM95], and uses a separate module to model both the psychological and physiological state of an agent. The high-level layer determines which goal the agent wishes to move towards (which then acts as an attractor force), decision making, memory, and orientation abilities. The low-level layer is responsible for collision detection and avoidance, with detection being performed on all overlapping objects and avoidance being performed only on objects in the direction of movement. Additionally, the low-level state of an agent also impacts aspects such as speed, fall probability, and pushing behaviours.

HiDAC has since been extended with personality models from psychology to represent the agents' psychological state. The work of Durupinar et al. [DAPB08] is one such example, where they use the popular OCEAN personality model [Wig96] for the psychological state of the agents. This model separates personality into 5 orthogonal dimensions of

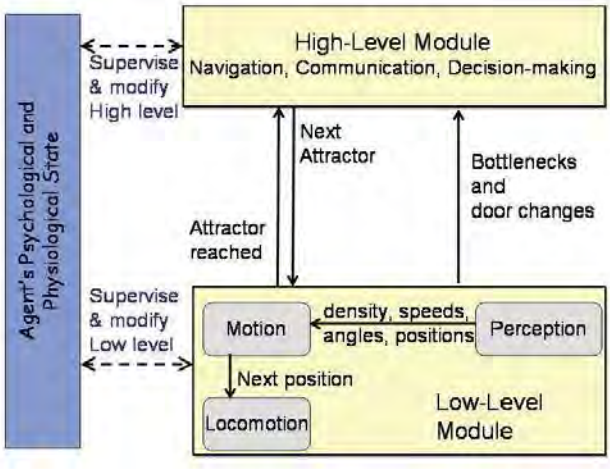

Figure 2.6: *Architecture for the HiDAC system, showing the interactions between the two tiers and the psychological state of an agent (Source: [PAB07]).*

personality space, namely *openness*, *conscientiousness*, *extroversion*, *agreeableness*, and *neuroticism*. The variables of this model are then mapped to specific functions of agents, such as leadership, sociability, patience, pushing behaviours, walking speed, panic and so on.

Another well known system that models agent psychological state is Musse and Thalmann's [RMT01] hierarchical ViCrowd system. The hierarchy of this system is separated into three levels: crowd, group, and individual. Parameters set at higher levels of the crowd are propagated to the lower levels, with the lower-level parameters also being adjustable, overriding the inherited parameters. This allows for groups or individual agents to specialize. For example, the crowd mental state can be defined as happy, with one specific group of agents as sad. An additional benefit of the hierarchical structure is that it allows for optimizing the system for real-time simulation. Musse and Thalmann achieve this by having group members share the same decision process.

Guy et al. have also proposed various crowd simulation models that take agent psychology into account, including PLEdestrians [GCC+10], based on the theory that living organisms naturally move along the paths that requires the least effort, as well as using the Eysenck 3-Factor personality model [Eys92] to define different personalities for agents in order to achieve heterogeneous agent behaviours within crowds [GKLM11].

A more recent work by Kim et al. [KGML12] aims to adapt the psychological state of agents during the simulation. They achieve this by modelling the stress that an agent receives, which gets mapped to their aggressiveness and impulsiveness using Seyle's [Sel56] General Adaptive Syndrome. This impulsiveness and aggressiveness are combined with a statically defined personality for the agent in order to determine the agent's final behaviour.

A different approach to modelling psychological aspects of agents appears in the work of Yeh et al. [YCP+08], who use *composite agents* to model aspects such as social priority, aggression, authority, protection, and guidance. Composite agents are normal agents in a crowd with a set of proxy agents attached to them. These proxy agents are not visualized, however, the other agents within the simulation nevertheless react to them. A simple use case for composite agents comes in the form of modelling aggression (figure 2.7), which is

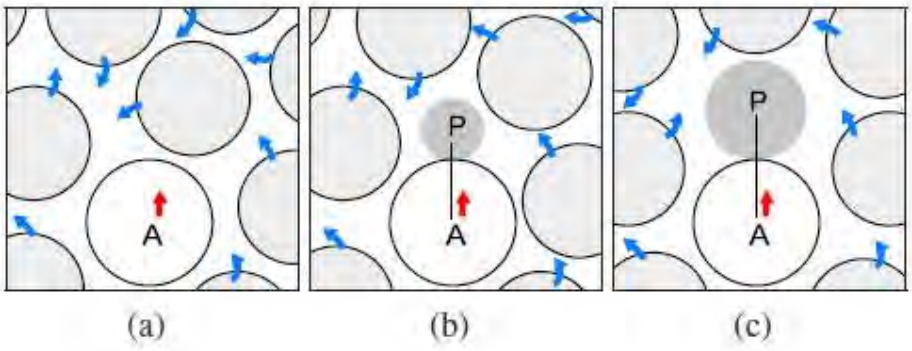

Figure 2.7: *Composite agents can be used to model aggressiveness in an agent by adding a proxy agent* p *in front of the direction of movement of agent* A*. Larger proxy agents take up more area and thus signify higher aggression (Source: [YCP+08]).*

achieved by attaching a proxy agent in front of a normal agent. This will result in other agents opening a path in front of the composite agent, which emulates how aggressive individuals intimidate others.

Data-driven models for controlling agents have also received a lot of recent attention [CC07, LCHL07, JCP+10]. These techniques typically aim to capture and store behaviours of real-life crowds in state-action pairs. Agents are then able to query these during simulation in order to determine what action is desired in a given situation. Due to the data-driven nature of these systems, they are more flexible than many of the other methods, as they are not limited to simulating one specific type of agent. Additionally, as the state-action pairs are obtained from footage of real-life organisms, the behaviours created are much more realistic.

The drawbacks to these systems are that it is not always possible to obtain the relevant behavioural data, and because computer vision techniques are used to extract the state-action pairs, some of the information may be inaccurate. Additionally, these methods are also often more computationally expensive than others, resulting in difficulties in simulating dense crowds.

In addition to modelling an agent's behaviour, it is also important to model how they interact with their environment. A problem here is that different parts of the environment require different interactions. For example, a human interacts very differently with a swimming pool than with a bench, requiring that each agent have a very large set of behaviours and rules in order to account for all possible objects in the environment. A different approach to this problem was introduced by Sung et al. [SGC04], who instead store the behavioural information within various environment structures, termed *situations*, and temporarily add this behaviour to interacting agents by appending to their motion graph the relevant animations (figure 2.8). This technique is advantageous as it allows for both a more scalable crowd simulation system, as well as the composition of different situations, allowing for complex agent interactions with multiple situations.

There have also been systems that, instead of providing a predefined behavioural model, provide a framework that allows users to more easily define them. One such system achieves this by using timed stochastic parametric conditional Lindenmayer systems (L-systems) [NTT92, NT93, NT96], a formal grammar that expands a set of symbols into a larger set of symbols through the use of some production rules. In these systems, the user

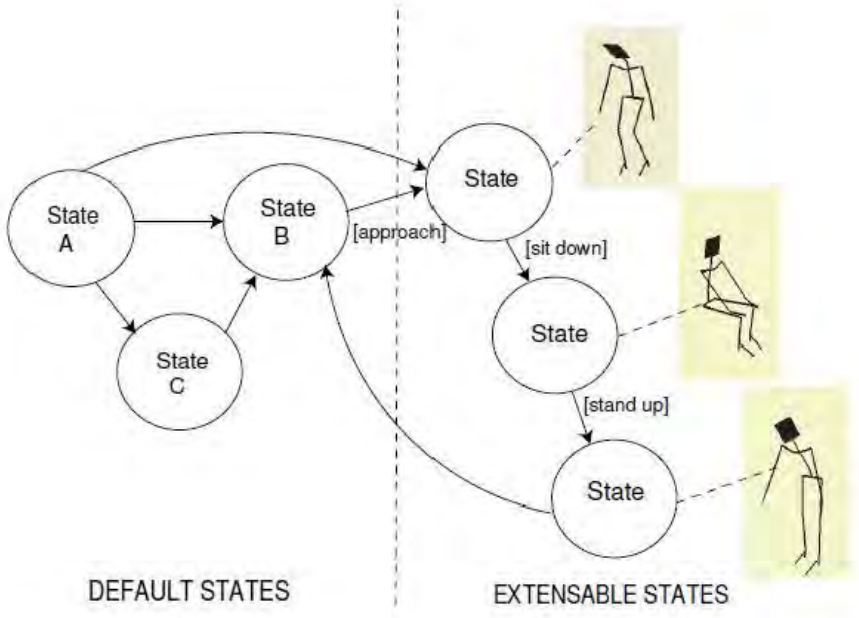

Figure 2.8: *In order to temporarily add a situation's behaviour to an agent in the method provided by Sung et al., a sub-graph detailing this new behaviour is appended to the motion graph of the agent at the relevant position (Source: [SGC04]).*

defines the behaviour of the agents with a set of production rules. During simulation, the agents' environment view along with the current time step is passed to the L-system, which then determines the performed behaviour and expected position of the agent by taking into account the production rules and by differentiating the time-step.

Another system that allows for the user to create their own behavioural models is *Massive*. This is achieved by allowing users to define the structures of the fuzzy systems that act as agent controllers. Massive will be discussed in more detail in section 2.6.

2.2.3 Embodied agent systems

All the crowd simulation techniques discussed so far represent agents within the environment as points, with the behavioural controllers determining how an agent should act and move. The animations in these systems are created prior to the actual execution of the simulation, and are merely played when the corresponding action is performed. Although this approach produces convincing results, there are often cases where the animations do not perfectly align to reality, leading to a break in immersion. A simple example of this would be an agent attacking another, where the animations for striking and blocking do not align properly.

Embodied agent systems aim to resolve this by treating the agent as a full-bodied system rather than a point in space. The animations for agents in such systems are not key-framed and assigned to actions prior to the simulation, but rather performed by modifying the positions and angles of the joints during simulation. This allows for a much wider variety of possible animations, resulting in more believable agents.

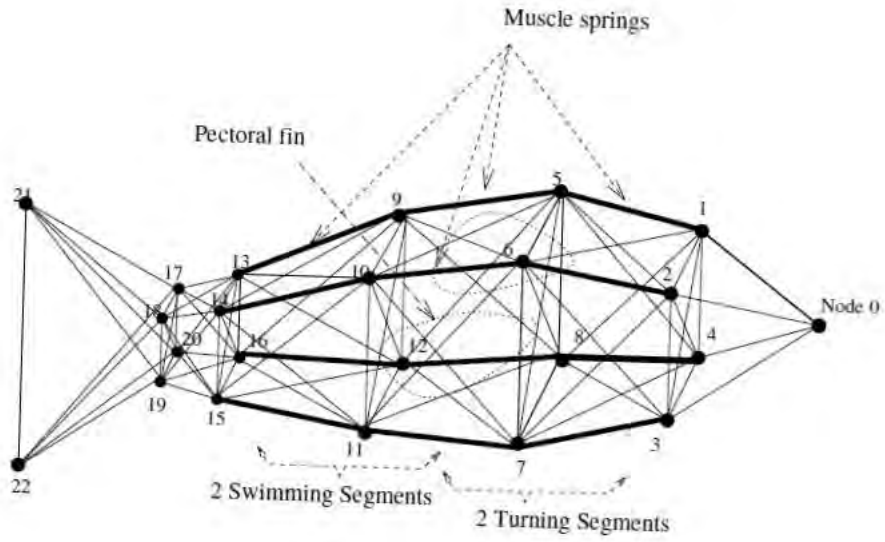

Figure 2.9: *The spring-mass system of an artificial fish with the springs at their rest lengths (Source: [TT94]).*

An early example of this type of system comes from Tu and Terzopoulos [TT94], who use an embodied system to animate schools of fish. A fish is simulated by separating its system into 3 subsystems, namely *Perception*, *Decision*, and *Action*. The perception subsystem is responsible for providing environmental information to the fish. This is achieved by incorporating two sensors into the fish. The temperature sensor is responsible for determining the water temperature at the centre of the fish, and the vision sensor is responsible for conveying information such as the colour, size, distance, and identity of objects within a 300 degree spherical volume.

The action subsystem deals with the low level motor action. The fish is modelled as a spring-mass system (figure 2.9), and moves by swishing its tail through the contraction of its muscle springs, which causes a reactionary force proportional the the amount of water displaced, propelling the fish forward. This provides realistic movement animation for the fish, closely resembling how real fish move.

The decision subsystem is responsible for sending orders to the action subsystem so that the fish knows which action to perform. It does this by sending commands in order to fulfil the current goal of the fish, which is determined by taking into account the fish's perception, habits, and mental state. By modifying the habits of various fish, one is also able to create more heterogeneous behaviours for the fish in the school. This system showed very promising results, with the school of fish achieving a variety of interesting emergent behaviours with minimal intervention by animators.

Another investigation into embodied agent systems was performed by Brogans and Hodgins [BH97]. They compare three crowds with differing body dynamics: a crowd of one legged robots; a crowd of cyclists (figure 2.10); and a crowd of points with masses. In their system, the rule set provided to the agents was similar to that of boids, however, as the embodied agents have to take into account the mechanics of the agent bodies, they are not guaranteed to reach the desired position and orientation specified by the

controller. The authors found that with embodied agents, the movement and animation quality of the crowd improved significantly compared to the point mass crowds. However, the complex bodies of the cyclist agents caused them to struggle with many tasks and made them difficult to control.

Despite the initial embodied agent systems demonstrating high-quality crowd animations and interesting emergent crowd behaviours, there has been a lack of subsequent work in this area. This could be attributed to the expense of simulating such systems, resulting in them being only really viable for simulating smaller crowds. Additionally, the level of realism achieved by these techniques is not necessary in many fields. Another issue with these systems is the difficulty of controlling the agents. This is especially problematic in film as the crowd and agents are often directed to perform specific tasks in order to enhance story.

Despite not gaining much traction in the crowd simulation domain, there has been a substantial amount of work in the area of single agent simulation and computational intelligence, with some recent examples including a neural network to train an embodied agent to perform bicycle stunts [TGLT14], and training bipedal agents to walk with a muscle-based control system [GvdPvdS13].

2.3 Macroscopic Crowd Simulations

An alternative to microscopic approaches is to simulate the crowds on a group level. These techniques are termed group-based [TGMY09, RMT01], top-down [MOS09], or macroscopic [PPD07, NGCL09] crowd simulations.

2.3.1 Flow-based methods

A traditional view of crowds would be as a collection of agents, with each agent in the crowd having a separate identity that interacts with other agents in order to achieve some type of aggregate behaviour. A different approach is to instead treat these crowds as flows. Hughes [Hug03] argues that this is possible as crowds are similar to fluids, and not irrational and erratic as was originally believed. In Hughes' work, he models crowds with fluid dynamics, and then analyses their properties. Unfortunately, he did not carry this through to create a simulation for these crowds. Nevertheless his work allowed a variety of flow-based crowd simulation methods to be developed over the course of the next decade.

One such system is Continuum Crowds by Treuille et al. [TCP06]. In continuum crowds, agents within the simulation are divided into groups. For each group, a dynamic potential field is generated by integrating both the global navigation and the obstacles of that group, resulting in a map detailing the desired (low potential) and undesired (high potential) areas of the environment for the group of agents. In order for the agents to find the ideal next position, they merely need to move opposite to the gradient of this potential field (as shown in figure 2.11).

In addition to continuum crowds, other well known methods that aim to use flow to direct agent movement include using velocity fields to direct the movement of agents [Che04, PVDBC+11], and using fluid dynamics equations for local collision avoidance

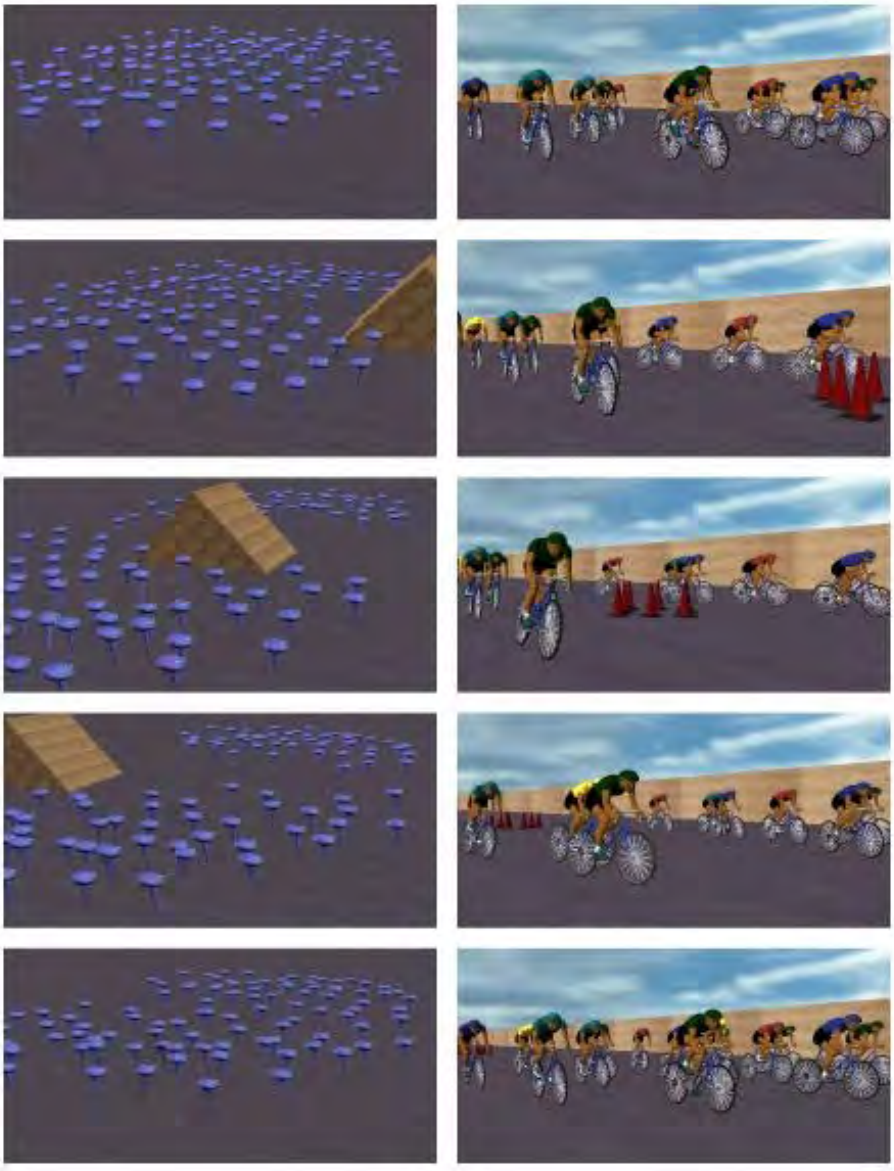

Figure 2.10: *Embodied agents in the form of one legged robots and cyclists, which were then compared to point mass agents (Source: [BH97]).*

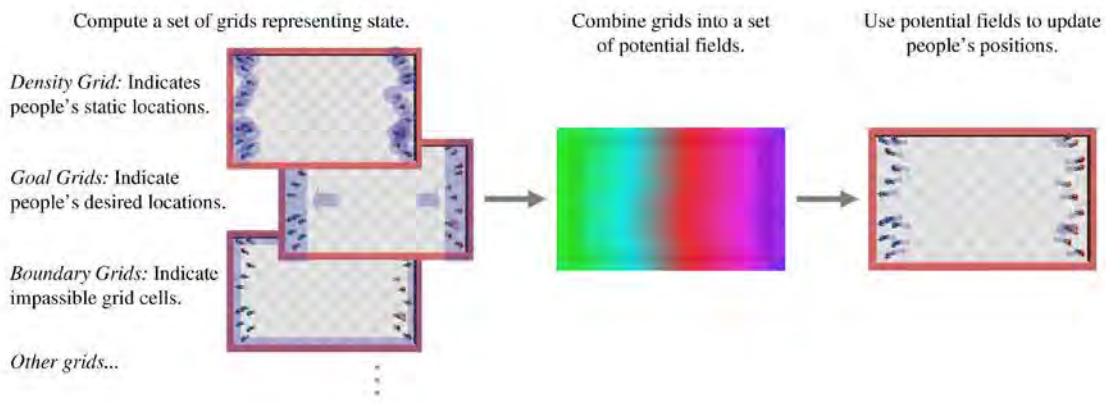

Figure 2.11: *The simulation process of the continuum crowds system (Source: [TCP06]).*

in dense crowds [NGCL09] in order to de-couple the computational expense with the number of agents.

2.3.2 Macroscopic Data-driven methods

Another macroscopic approach to simulating crowds is to stitch or synthesize precomputed or captured behaviours, in order to achieve some desired output behaviour. Group Motion Graphs [LCF05] is one such technique. In Group Motion Graphs, a series of crowd motion clips are clustered and stitched together in a graph. In order to generate crowd motion, this graph is then traversed. A problem with this method is that it has difficulty with agents that have significant body dynamics. This could cause problems with linking clusters as the correct agent orientation, position, and body state are unlikely to always be found. Another problem with this technique is that it cannot deal with the removal or addition of agents to the simulation.

A different data-driven technique is investigated by Yersin et al. [YMPT09], who create a large environment with a series of small environmental blocks. Each environmental block has a set of agent paths defined. These blocks can be then stitched together (figure 2.12) to create a larger environment in such a way that agent paths are aligned with adjacent blocks, allowing agents to move from one block to another.

Although the above techniques allow for the real-time simulation of large groups of agents in a variety of different environments, they require a pre-processing step. Additionally, they are somewhat rigid as it is not possible to simulate what has not been catered for prior to the simulation.

2.3.3 Pathfinding

It is often required in crowd simulations that agents move from one location to another. While not a full crowd simulation system, pathfinding can be viewed as a macroscopic approach as it aims to plan out the agents' paths, generally on a group level, by taking into account the global environment.

In contrast to more traditional path planning, finding the global path is not visually sufficient for crowd simulations, as it is important to take into account local collision

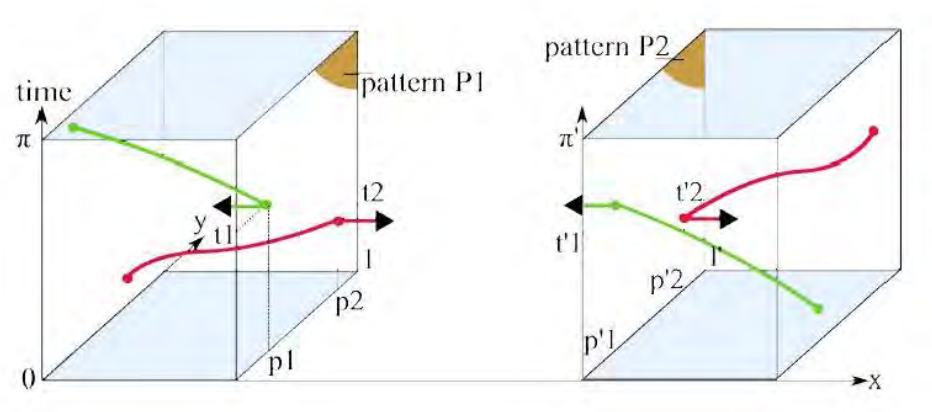

Figure 2.12: *Two blocks with paths that can link up to form a larger block. Agent paths are specified in red and green, with the red directing flow to the right and the green directing flow to the left (Source: [YMPT09]).*

avoidance and deviations in different agents' paths. This has resulted in substantial research investigating path planning in this context [KO04, FSN09, KGO09, BLA02, Ger10, GO07].

Most path planning methods for crowd simulations approach the problem in a similar manner. First, the roadmap of the environment is generated. This can be done either by artists defining some navigation mesh, or by using techniques such as probabilistic roadmaps [KSLO96], Voronoi-based methods [For92], or rapidly exploring random trees [LaV98]. From these roadmaps, a traditional pathfinding algorithm such as Dijkstra's shortest path [Dij59] or A* [HNR68] can be used to calculate the global path to reach the goal. After the path has been constructed, it is then expanded in width to represent a corridor instead of a line. These corridors are expanded in such a way that they do not contain any environmental obstacles. Agents are then guided along these corridors towards the goal, with some local collision avoidance model used so that they do not collide, and that the agents adopt different trajectories. An example of a fully planned out corridor is shown in figure 2.13.

2.4 Additional Aspects to Crowd Simulation

Previous sections describe a variety of behaviour models for crowd simulations. However, there are many additional aspects that need to be considered in order to implement a full crowd simulation system. As we are primarily interested in the behavioural models, we will not describe these additional areas in depth. Rather, this section aims to provide a general high-level overview.

2.4.1 Modelling

Modelling refers to defining the geometric shapes that make up the agents and the environment within an animated scene. These modelling methods can be divided into three categories: creative, reconstructive, and interpolated [MTSC04]. Creative methods require an artist to define the geometric shapes through either subdivision modelling or

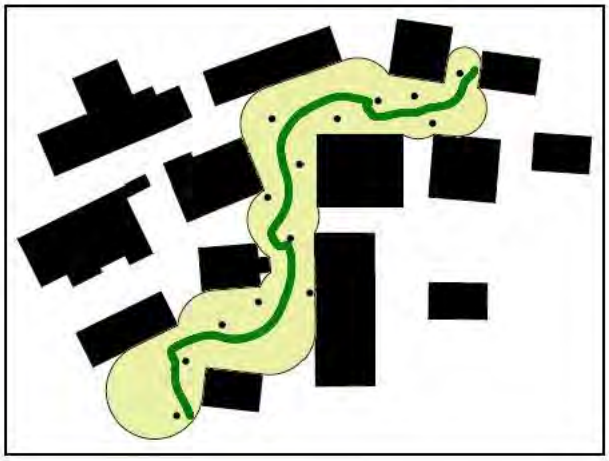

Figure 2.13: *Motion planning for crowds. The green line is first calculated which finds the path to the destination, after which a corridor is expanded around it (Source: [KGO09]).*

through patches, such as NURBS or Bezier curves. These methods are unfortunately often infeasible for crowd simulations due to the complexity and time required to model large crowds.

Reconstructive methods aim to capture the geometry of existing objects. This can be typically done by using 3D scanners, images, or video sequences [Tha07]. Although these techniques perform well, as they provide both realistic geometry and animation for agents, they come at the expense of considerable computation. Additionally, the systems and equipment used in these techniques are often expensive, making them infeasible for most crowd simulations. Another problem is that it is rather difficult to modify or augment these meshes according to a user's intentions, limiting the storytelling freedom of artists [Tha07].

Interpolated methods aim to blend existing models together in order to create new ones. These techniques have seen much success, and have been used to produce background characters for films such as *Shrek* [Tha07]. However, they do not extend well to main characters as one often needs to model specific details. Thus, these main characters are typically modelled with either creative or reconstructive methods, or acted out by real-life actors, who are then superimposed into the virtual environment.

2.4.2 Authoring

An easy-to-use authoring system is also necessary to produce content for a crowd simulation system. One way to author these crowds is to provide a series of constraints and crowd configurations, and have the agents attempt to adhere to them. This approach has been used in the works of Anderson et al. [AMC03] and Jacka [Jac09]. Another intuitive method of authoring crowds is to use a painting interface, where the user can specify, modify, and guide agents interactively by drawing various shapes depicting goals and guiding flows. This approach has been widely used by Ulicny et al. [UCT04], Sung et al. [SGC04], and Patil et al. [PVDBC+11].

Additionally, there are also systems that allow for the easy authoring of agent behavioural models. *Massive* is one such system, and allows for this by providing an inter-

face that enables the user to join up various nodes in a directed graph in order to easily define the rule-set for an agent's behavioural model.

2.4.3 Rendering

Virtual environments are typically rendered by processing every single geometric object in the environment on the GPU. This quickly becomes a problem with large crowds of agents as the geometry being processed becomes infeasibly large. As a consequence, a series of optimizations are needed to feasibly render these crowds.

Levels of Detail (LoD) [Lue03] is a concept that has been extensively researched in computer graphics. In LoD, objects that are closer to the camera are defined in more detail, as their flaws can be spotted more easily. This can be used in the rendering of crowds to improve efficiency, because only the agents that are close to the camera need to be rendered in full detail. Agents that are far away can then be rendered using either *Imposters*, which are quads with a texture representing the agent [TLC02, DHOO05], or by pre-baking the agent animations on the CPU, which allows for faster display as skeletal deformations have already been pre-calculated.

In addition, standard rendering optimizations approaches used for optimizing rendering of normal scenes such as occlusion and frustum culling are also very useful for rendering crowds [Tha07].

2.4.4 Behavioural Levels of Detail

In addition to performing LoD optimizations for rendering, these techniques can also be adapted to the behavioural models of the agents. This can be done by simulating only the behaviour of the visible characters [Bro02]. However, this has the drawback of the simulation being incoherent, for example looking away and then back at an area can result in agents being in unexpected positions, or different agents taking the place of the old. Due to this problem, behavioural LoD is far more complicated to handle than rendering.

A differing LoD approach for behaviour was proposed by Thalmann [Tha07] where the agents behaviour were separated into various levels, with agents farther away receiving a lower level of simulation than the agents close by. These levels were set up in such a way that the agents could transition smoothly from one level to another, thus reducing visual artefacts.

A hybrid model that incorporates both Microscopic and Macroscopic techniques has also been used to this effect in order to simulate large-scale traffic [SWL11]. In this method, the environment is divided up into regions. Regions that are close are simulated using a Microscopic behavioural model, and regions that are far are simulated using a Macroscopic one. In order to convert these regions from a Microscopic representation to a Macroscopic one, averaging is used, whereas to perform the opposite, a Poisson-like process is used. In order to account for traffic flowing between regions, the agents are instantiated and disappear according to the flux calculated between the respective regions. The major problem with this method is that there is no coherency between the different models, thus alternating them at a fast pace results in a very incoherent scene.

2.4.5 Collision Avoidance

Collision avoidance is one aspect that all crowd simulation systems need to consider. It is important as it often allows one to eliminate the use of collision detection, which is computationally expensive, and allows agents to adopt more realistic trajectories. Although some techniques implicitly handle this [TCP06, Che04], others need to define explicit collision avoidance for the agents.

Although Social Forces [HM95] can be treated as a full behavioural crowd simulation system, it can also be used purely as a collision avoidance model. However, it has drawbacks such as the possibility of agents getting stuck if the forces cancel each other out, and agents sometimes adopting oscillatory behaviours. There has been strong recent work on collision avoidance for multi-agent systems which deals with many of the issues of the Social Forces model, with some popular algorithms being velocity obstacles [FS98], reciprocal velocity obstacles [VdBLM08], and vision-based collision avoidance [OPOD10].

2.4.6 Directability

It is important in certain application areas that the crowd be directable. This refers to controlling the behaviour of both specific individuals, as well as the crowd as a whole. As controlling the behaviour of crowds is our primary interest, this area will be discussed in depth in section 2.5.

2.5 Controlling Crowd and Agent Behaviour

One key requirement for crowd simulations in fields such as film animation, is that the crowds be directable. The reason for this is that emergent behaviour is often not sufficient for storytelling, resulting in severe restrictions to the artistic freedom of the writers and artists.

As a result, there have been a variety of research on directing the behaviour of crowds. These separate crowd control into two different types: *High-level* and *Low-level* control.

2.5.1 Low-Level Control

Low-level control refers to directing the behaviours of select individual agents within the crowd. Although it is possible to instead key-frame these specific agents, it quickly becomes infeasible as the number of agents needing control rises. A more scalable approach is to allow modifications to the rule-set of those specific agents so that they behave as desired.

Blumberg and Galyean [BG95] introduce a method that allows users to achieve low-level control over agents on three different levels (figure 2.14). On the motivational level, the user is able to modify the high-level motivations of the agent. For example, changing the hunger levels of an agent from not hungry to very hungry will prompt the agent to search for more food. Modifications on this level can be viewed as a suggestion to the agents.

The task-level is responsible for telling an agent what to do, such as moving to a specific position in the environment. Because the user does not specify exactly how the

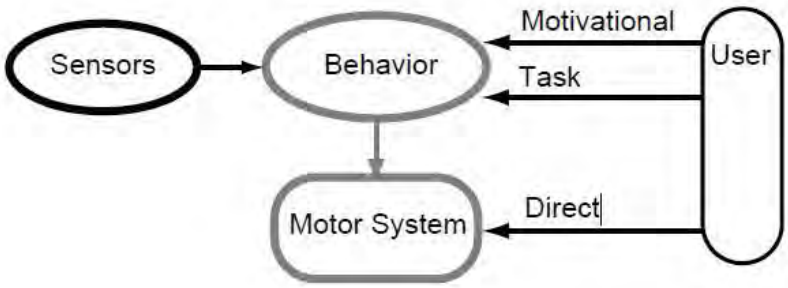

Figure 2.14: *Architecture for allowing multi-level control of agents. The motivational and task levels influence an agent's decision making process whereas the direct level influences the low-level motor functions of the agent (Source: [BG95]).*

agent should achieve this, the agent still bases its movement on pre-existing behavioural and environmental rules.

The lowest level of control is the direct-layer. In this layer, the user has direct control on the motor system of the agent, and can thus specify exactly how an agent should behave. An example of this would be telling a dog to wag its tail or to raise its left leg. In addition to low-level control, the layered approach adopted by this system also allows users to easily author this control, resulting in a very promising approach to controlling individuals in crowds.

A different approach, termed Cognitive Modelling, was proposed by Funge et al. [FTT99]. They use a cognitive layer to allow for control of individual agents. Artists specify the behaviours of agents as a set of logic statements within this cognitive layer, which is then expanded into a state tree. Subsequently, one can then choose a specific desired state for the agent, which then results in a tree search for this state with all the preceding required actions and states being found in the process.

An advantage of this technique is that as long as a state exists, it is possible for the agent to satisfy it, allowing for very precise control over the agents. Additionally, it is possible to animate other objects in the scene with this technique, such as cameras and lights. However, it can be very computationally and memory intensive (for more complex agent behaviours), as the state tree would grow significantly. A similar technique comes from the works of Lau and Kuffner [LK06], who use precomputed search trees containing motion clips for online motion planning of agents. This technique is shown to be very effective, allowing for the path planning of roughly 150 agents in a real-time and dynamic environment.

In addition to the above, it should be mentioned that systems that aim to simulate the psychology of agents [PAB07, DAPB08, KGML12] can all control the behaviors of single agents to varying degrees by merely modifying the psychological states or personalities of these agents. In contrast to these systems, Macroscopic techniques have particular difficulty in enforcing low-level control of crowds. This is primarily because they treat the crowd on a group level rather than an individual one.

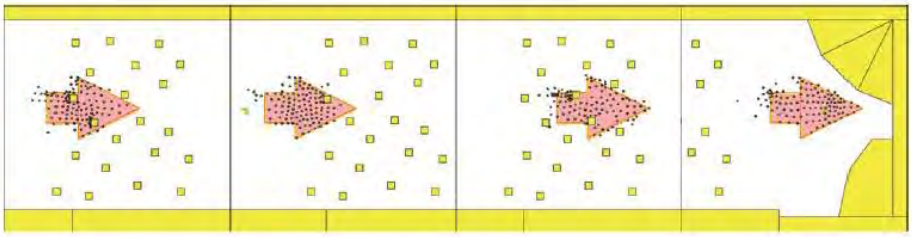

Figure 2.15: *Snapshots of a crowd moving through an environment scattered with obstacles. The agents evade obstacles when they are in close proximity, otherwise they aim to adhere to the arrow shaped formation (Source: [CL08]).*

2.5.2 High-Level Control

In contrast to low-level crowd control, high-level control refers to control of the aggregate behaviour of crowds. Examples of this include controlling crowd formations, controlling crowd movement, and determining outcomes of battles. One way of obtaining this control would be to modify the parameters of agents until the desired crowd behaviour emerges. However, this is a rather tedious task as it requires a significant amount of trial and error, as well as expert knowledge. Thus, methods for automating this type of control are necessary.

Anderson et al. [AMC03] introduce just such an approach for controlling the movement of bird flocks using a variant of Reynold's flocking model [Rey87]. In their system, the user defines a set of constraints for the bird flock, such as requiring an agent to pass through a specific point, the center of mass of the flock to pass through a point, or having the flock lie within some shape. The algorithm then proceeds to run an unconstrained version of the simulation, as well as a set of simulations that satisfy the specified constraints. They then compare the set of constrained simulations with the unconstrained one. The constrained model that is most similar to the unconstrained one is then used as the final simulation.

This technique is interesting in that it allows the flock behaviour to satisfy hard constraints whilst retaining much of its realism. However, it is specific to bird flocking due to the behavioural model used. Another problem is that it can only be used for motion control, and does not deal with other types of high-level control, such as specifying battle outcomes or determining the number of surviving agents after some emergency.

The shape control constraint in the work of Anderson et al. [AMC03] has since been further investigated in more recent work. Chang and Li [CL08] create a shape template system coupled with the use of Fuzzy Systems [BT97] for the agents brains. Fuzzy systems use *Fuzzy Sets*, which are based on set logic that takes into account degrees of membership. This is advantageous as it allows agents to perform actions to varying strengths. These *Fuzzy Sets* are represented as *Fuzzy Nodes* within a directed graph, and receive input information from either the environment, or from other Fuzzy Nodes, and produce an output based on rules pre-specified by the designer of the system. Their technique allows the crowd to attempt to adhere to a user specified formation whilst taking into account other agents and the environment (as seen in figure 2.15).

In contrast, Ho et al. [HNOC10] use a weighted sum of forces approach. Here, the force of an agent is the combined weighted sum of an obstacle avoidance force, and a

formation adherence force. When agents move closer to obstacles, the weight of the obstacle avoidance force grows larger, forcing them to break from formation. After they have passed the obstacle, the weight of the obstacle avoidance force becomes smaller, allowing the formation adherence force to move the agents back towards their respective positions.

A more recent approach to high-level control in crowds is provided by Jacka [Jac09]. Much like both *Massive* and Chang and Li's method, the agent controllers are defined using Fuzzy Systems. In order to control the aggregate behaviour of the crowds, the membership functions of the fuzzy nodes are trained with particle swarm optimization [Ken10] until the behaviour of the crowd satisfies the user-specified behaviours. This approach was shown to work well across a variety of control tasks and agents. However, it does not guarantee the complete satisfaction of the constraints, and also requires expert knowledge in the modelling of fuzzy controllers, leading to possible difficulty in training the controllers if they are poorly defined.

In addition to the discussed systems, most macroscopic techniques are able to provide some degree of high-level control of crowds since they simulate crowds on a group level. However, as discussed in section 2.3, macroscopic crowds lack the believability that microscopic crowds provide, and are therefore infeasible for certain applications, such as film.

2.6 Crowd Simulations in Films

One major application area for crowd simulations is that of crowd shots in films. These crowd shots serve two major functions: enhancing the virtual environment, allowing for more immersiveness; and enhancing storytelling capabilities, allowing directors and writers more artistic freedom. In order to achieve these goals, the crowd simulation techniques used must allow for believable behaviours, high quality visualizations, a high degree of automation, easy to use authoring, efficient computation, and directability [Jac09, Tha07, THL$^+$04]. In addition to these requirements, crowd simulations in film differ from others in that scenes are not rendered in real-time and shots are short, with no need for shot to shot coherence [THL$^+$04].

Although many of the crowd simulation systems used by films are developed in-house [Tha07], there has been one commercial system that gained great popularity and widespread use within the industry. *Massive* is a microscopic system that allows for high-quality crowd simulations, and has been used in a wide variety of fields such as film, television, games, education, architecture, and engineering [Masc].

As with many of the discussed systems, Massive approaches simulating crowds in three steps: perception; decision; and action [Jac09]. In the perception subsystem, the agent reads input from its surrounding environment, and then provides these inputs to the decision subsystem. Massive provides a set of predefined senses, such as sound, vision, flow-field, collision, and colour [Lak07]. Most of these senses are self-explanatory, with perhaps the exception of flow-fields, which can be thought of as guiding forces in the environment to help an agent move towards a specific location. It should be noted that although flow-fields are a macroscopic concept, Massive itself is still a microscopic

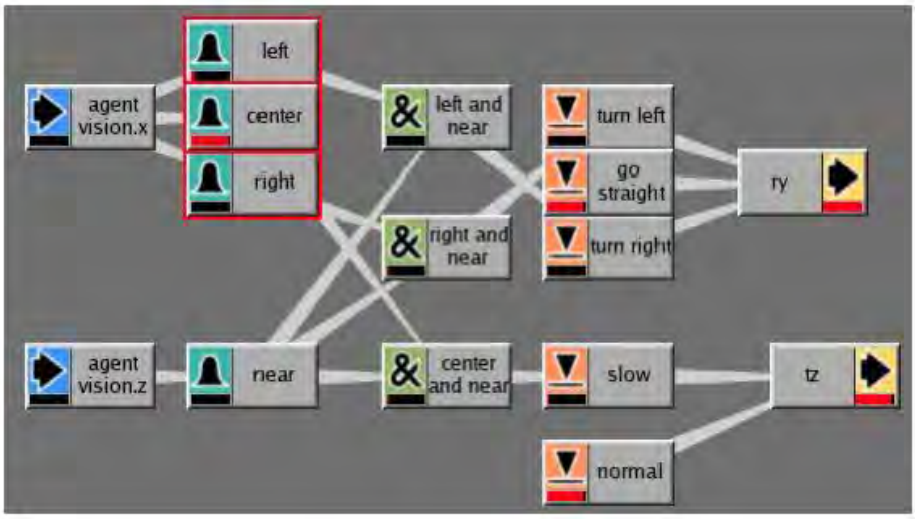

Figure 2.16: *An example of a brain in Massive, where the nodes represent either rules or Fuzzy Sets, and the edges represent information flow (Source: [Masa]).*

technique as these fields do not force the agents to move in the set direction, but merely represent a suggestion.

The decision subsystem determines how an agent should act based on its received inputs, using Fuzzy-Systems as the agent controllers. Figure 2.16 shows an example of a brain in Massive, where the edges depict information being transferred between nodes, and the nodes specify either rules, input sensors, or Fuzzy Sets. Unlike many of the previous systems, Massive itself does not contain any behavioural models, but rather acts as an easy-to-use authoring system for artists to define these models. However, Massive does provide some pre-made behavioural models for a variety of different agents [THL+04].

The action subsystem enables users to define a set of possible actions for agents, which are then connected up to the various output actions of the decision subsystem. These actions are connected together in a motion graph, which defines the sequences of actions that an agent can perform [Lak07]. Each of the actions defined is then linked to an animation, obtained either by traditional key-framing or motion capture [Lak07]. The action subsystem benefits substantially from the use of fuzzy systems in the decision subsystem, as the degree to which an agent performs an action helps the action subsystem blend the motions together.

One problem that Massive, along with other Microscopic techniques, suffers from is the issue of high-level control. In Massive, this is solved by having an expert modify the fuzzy system until the crowd behaves as desired. Unfortunately, this is a very time consuming task. Flow-fields are an alternative for dealing with this control problem. However, these fields do not act as an absolute command, allowing agents to ignore them, and they also only deal with motion control. Additionally, despite Massive providing an easy-to-use interface to define the fuzzy controllers, it still requires expert knowledge, enabling only a select few individuals to understand and create agent behavioural models. The interface for building fuzzy controllers becomes particularly problematic when more

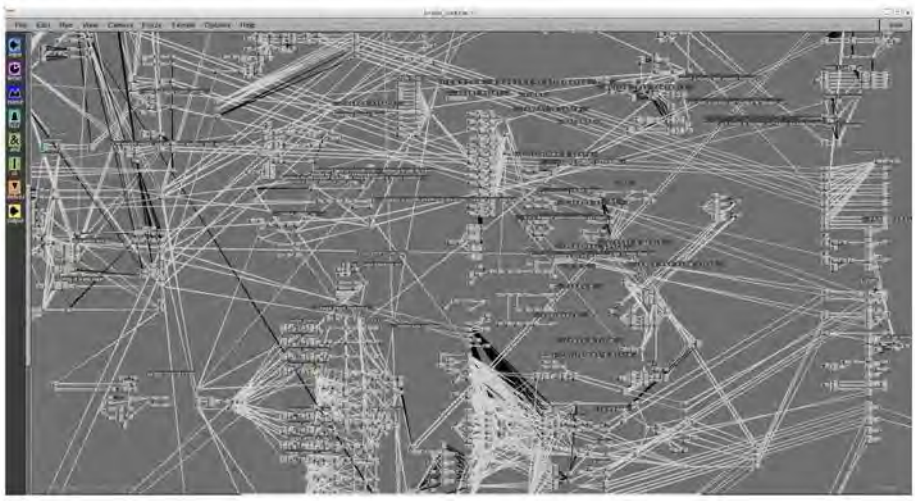

Figure 2.17: *Complex agent behaviours in Massive typically result in very complicated brains, and are usually very difficult to understand (Source: [Masb]).*

complex behaviours are required, as it leads to very large brains unreadable to all but the original creator (figure 2.17).

2.7 Summary

In this chapter we have discussed a wide variety of crowd simulation techniques. These techniques can be divided into five groups: Microscopic Cellular Automata models, Microscopic rule-based models, Microscopic embodied agent models, Macroscopic flow-based models, and Macroscopic data-driven models. These techniques all have their own strengths and weaknesses, and thus different types of techniques are preferred for different applications.

In order for a crowd simulation system to be useful for films, it needs to satisfy the requirements listed in section 2.6. Table 2.1 details how well the various classes of crowd simulation techniques satisfy these requirements, with a scale ranging from very low to very high.

Behavioural believability refers to how closely an agent's behavioural model mimics reality. Embodied agent systems generally display the most realistic behaviour as they simulate the full body system of agents, whereas flow-based techniques are the least convincing due to the overly homogeneous behaviours of the agents, leading to crowds that often appear more akin to particle systems than crowds of autonomous agents. Although believability is very important for films, this is only to the point that agent behaviours are sufficiently convincing.

Visualization quality refers to the quality of the animations and renders that can be generated by the crowd simulation systems. This is the most important requirement for film, as no matter how well a system satisfies the other requirements, the final product is still unrealistic if these aspects are poor. Embodied agent systems once again perform very well in this area, as the animations are generated during the simulation whilst taking into account the body dynamics of the agents. Cellular automata systems perform poorly

Table 2.1: *Satisfaction of the various requirements of crowd simulations by different classes of crowd simulation techniques. As Massive is somewhat of a unique system, it has been given its own category.*

	Macroscopic Techniques		**Microscopic Techniques**			
	Flow	**Data-driven**	**CA**	**Rule-based**	**Embodied**	**Massive**
Behaviour Believability	low	med	med	high	v-high	high
Visualization Quality	low	med	v-low	high	v-high	high
Automation	low	med	low	med	v-high	high
Authoring	high	v-low	med	med	low	med
Computational Efficiency	v-high	high	high	low	v-low	low
Directability	med	med	low	low	v-low	med

in this regard as it is difficult to create convincing animations for these systems due to their discrete nature.

One of the main benefits of using crowd simulation systems in films is that they automate much of the work that an artist would otherwise have to perform by hand. Thus, a crowd simulation that requires a lot of user input is far less useful than one that does not. Most categories of crowd simulation techniques automate most of the simulation process, however, some techniques receive a lower rating as they are unable to generate complex behaviours on their own, resulting in artists having to manually add animations to the characters.

In order for a system to be useful, it needs to be easy for artists to define the required crowd behaviours, and to make adjustments. Flow systems are very good at accomplishing this, as using painting interfaces with these techniques is very intuitive. In addition to defining the initial positions of agents, painting interfaces can also be used to interactively edit crowd motion by drawing flow-fields onto the environment. Data-driven techniques, on the other hand, are poor at authoring due to their data-driven nature. Changing a crowd's setup often requires new data, resulting in a large amount of work in order to modify the crowd behaviour.

Although films do not require real-time simulation, it is still important for the techniques to be efficient in order to decrease the render time. Flow-based systems are particularly efficient and can often simulate hundreds of thousands of agents in real-time. This is possible as they simulate the crowd at a group level, as opposed to other microscopic techniques, which aim to simulate the crowd at an individual level. Embodied systems, due to their complexity, received the lowest score in this category.

The final requirement for crowd simulations in film is directability. As was described in section 2.5, this can be separated into both high and low level control. Although embodied agent systems perform poorly in both - high level control due to their microscopic nature, and low level control due to the fact that they have to take body mechanics into account - other techniques generally excel in different types of control. For example, macroscopic

techniques allow for better high-level control, and microscopic techniques allow for better low level control. There are, however, few systems that are able to perform well for both types of control.

Since crowds in film do not need to be rendered in real time, the shots are short, and that there is no need for shot to shot coherency, it is possible to introduce high-level control to microscopic crowd simulations through the use of optimization techniques. This is achieved by using optimization techniques to modify the low level crowd behaviours until the desired behaviour emerges. Jacka [Jac09] approached the problem this way using Particle Swarm Optimization to optimize the membership functions of fuzzy controllers, which represented the brains of agents. However, this approach has the drawback that it is dependent on the defined structure of the fuzzy controller, as a poorly defined controller leads to difficulty in optimizing behaviours.

We aim to instead evolve Artificial Neural-Networks (ANNs) as the agent controllers. This has the benefit that no controller structure has to be defined beforehand, allowing for an easier authoring process. In the next chapter, we will provide an overview of ANNs, as well as provide promising approaches to training them.

Chapter 3

Neuro-Evolution

Neuro-Evolution (NE) is a paradigm for training Artificial Neural Networks (ANNs), which aims to evolve their structure or weights with Evolutionary Algorithms (EAs). This approach has been applied to a wide variety of tasks, and has been particularly successful in the domain of automating the design of agent controllers [TGLT14, PB09, TT12, YM09, GM99]. Due to the plethora of literature that use NE for automated agent controller design, it is attractive for us to investigate its merits when used in the context of applying control to crowd simulations.

3.1 Artificial Neural Networks

Self driving cars [GKG96], speech recognition [Lip89], image recognition [LMD+11] and many other important applications all utilize ANNs, a powerful computational model inspired by the biological brains of animals [Yeg09].

The base building block of an ANN is the *Artificial Neuron* (figure 3.1), a node that takes in a set of inputs and produces a single output. The output is typically obtained by passing the weighted sum of the inputs through an *activation function*. These *neurons* are connected together in a directed graph to form an ANN.

The *Activation functions* in the *neurons* map the weighted combination of the inputs to an output. Examples of *activation functions* include linear, sigmoid, and tan functions.

ANNs are desirable for machine learning because each neuron contains weights corresponding to its input signals. This allows for one to adjust the weights of the neurons within the ANN until the desired outputs are achieved. The adjusting of these weights

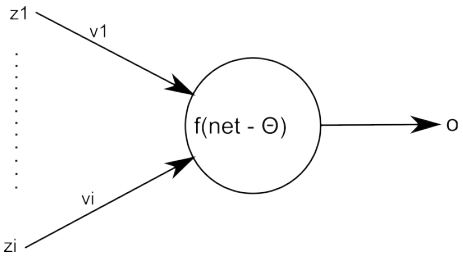

Figure 3.1: *An Artificial Neuron. The inputs $z_1..zi$ are combined together with the weights $v_1..vi$, with the net value passed to the activation function f after subtracting the bias θ.*

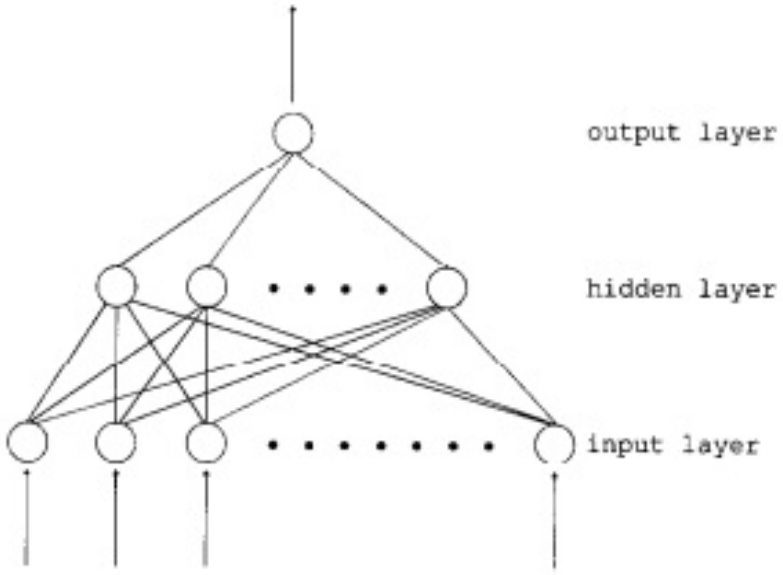

Figure 3.2: *A simple FFNN with 3 hidden layers (Source: [SKP97]).*

Table 3.1: *Truth table of the XOR bitwise operator*

x_1	x_2	$x_1 \oplus x_2$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

can be done either by hand, in the case of very simple problems, or via various training algorithms, which are discussed in section 3.1.2.

3.1.1 Feed-Forward Neural Networks

Feed-Forward Neural Networks (FFNNs) [RHW88] are a type of ANN where neurons are connected together in a non-cyclical directed graph, with information only being propagated forward, as seen in figure 3.2. The neurons in a FFNN are typically classified as either *hidden* or *output* neurons, where *hidden* neurons propagate information to other neurons and *output* neurons provide information representing the output of the ANN. These neurons are typically structured in layers, with each layer of neurons propagating information to the next.

FFNNs can be divided up into two categories, *Single* and *Multi* Layer Perceptrons [Lip87]. In a *Single-Layer Perceptron* (SLP), the inputs are mapped directly to the outputs of the network, with no hidden layer in-between. These networks have the major drawback that they are only able to learn linearly separable data patterns, which is data that can be separated by some hyperplane. A popular example of a problem that these networks are unable to solve is the XOR operator, a bitwise operator with the inputs and outputs shown in table 3.1.

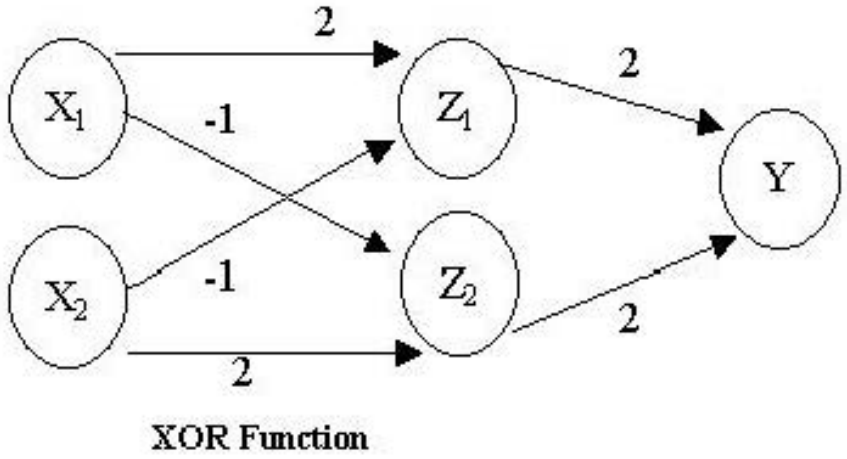

Figure 3.3: *ANN to solve the XOR operator (Source: [XOR]).*

The *Multi-Layer Perceptron* (MLP) is a FFNN with one or more hidden layers. Unlike SLPs, MLPs are capable of learning data that is not linearly separable, for example the MLP solution to the XOR operator is shown is figure 3.3. MLPs are the primary ANNs of interest to us in this thesis since single hidden layer MLPs are capable of universally approximating any continuous function on compact subsets of $\mathbb{R}^n$ for a specific set of activation functions (for example sigmoid) [Cyb89]. We are primarily aiming to use these MLPs as agent controllers within our simulations, where they determine the behavior of our agents, given a specific set of inputs representing the agents' view of the world.

3.1.2 Training ANNs

Despite the advantages of ANNs, it is often infeasible to design them by hand for non-trivial problems, as the networks for these problems are often large and difficult to understand. However, since the weights of ANNs are adjustable, their design can be easily formulated as a mathematical optimization problem, with the quality of the ANNs generated being quantified by a fitness value, which is typically determined by some task specific *fitness function*. This section introduces some of the more popular ANN training algorithms.

Backpropagation

Backpropagation [RHW88] is the most fundamental and well known algorithm for training FFNNs. In backpropagation, an ANN is provided with a training set, which is a sample of inputs and their respective expected outputs. The ANN is then fed these inputs, and their outputs are compared to the expected outputs. A function of the difference (error) between these outputs is then passed back to the ANN, and the weights are then adjusted in order to minimize these errors, essentially performing a non-linear regression of the training set.

Despite the popularity of backpropagation, there are some key drawbacks:

- Like other supervised training methods, it requires a training set, and therefore is not feasible for problems where the expected outputs are unknown or difficult to obtain;
- It is a gradient search technique. Thus, it cannot handle discontinuous fitness landscapes and also tends to get stuck in local optima [MD89]; and
- It typically requires an initialization pass in order to achieve good training results for ANNs with deep architectures (having multiple hidden layers) [LBLL09].

Additionally, one has to take the quality of the training set into account, as a poorly sampled training set will result in the network being unable to generalize to certain inputs. Another consideration when using backpropagation (as well as other supervised methods) is overfitting, which is when the ANN has memorized the training set, but has not learned to generalize to new data. Causes for overfitting include a sample set being either too small or too large, too noisy, training for too long, and the model having too many degrees of freedom. Methods of dealing with overfitting include obtaining larger samples and optimizing network structure [Eng07].

Deep Learning

Despite the fact that single hidden layer ANNs are able to universally approximate any mathematical function, it is still sometimes desirable to have ANN architectures with more than one hidden layer. One benefit that deep architectures provide over shallow ones is efficiency, as circuit complexity theory suggests that deep architectures can be more efficient than their shallow counterparts, as most functions representable by a compact deep network would require a large number of components in order to be represented by a shallow one [BL07]. Deep architectures are also useful for learning representations of data, where each layer of neurons is viewed as a representation of their inputs.

Despite the advantages of deep architectures, more traditional learning algorithms, such as backpropagation, struggle to train these networks. This can be attributed to the fact that with gradient search techniques, using large initial weights often results in the algorithm getting stuck in poor local minima, whereas using small initial weights results in very slow training speeds because the error signals propagated become smaller with each layer [HS06]. Furthermore, due to the additional layers, it may also be more computationally expensive to train networks with deep architectures.

Due to the difficulty in training these networks, a new set of algorithms are needed, referred to as *Deep Learning* algorithms. Delving into the details of these algorithms falls beyond the scope of this thesis. However, should the reader be interested in these algorithms, we recommend the work of Hinton and Salakhutdinov [HS06] as a starting point.

Deep learning has been used for a variety of challenging tasks, such as self driving cars [HES+08] and speech recognition [HDY+12, DYDA12]. However, for this project we have decided to not use it, as the networks that we use are generally simple enough to be trained by other algorithms, and more work in automated controller design has been conducted in other fields, such as Neuro-Evolution.

Meta-heuristics

Meta-heuristics refers to a class of search methods that are aimed at finding approximate solutions for complex optimization problems where more classical methods fail to be effective or efficient. As training ANNs can be viewed as a search for their ideal weights and topology, this class of algorithms can also be applied to accomplish such. Meta-heuristics can be defined as iterative generational processes which aim to use various strategies to achieve a balance between exploitation and exploration of a search space [OL96].

Meta-heuristic techniques can be classified into two groups, *single solution* and *population-based*. Single solution techniques aim to iteratively improve a single instance of the candidate solution, whereas population-based approaches aim to simultaneously improve many candidate solutions by typically using various population-based characteristics to guide this improvement. Examples of single solution methods are Simulated Annealing [AK88] and Iterated Local Search [LMS01], and examples of population-based methods include Particle Swarm Optimization (PSO) [EK95], Genetic Algorithms (GAs) [Hol92], and Differential Evolution (DE) [SP95].

Meta-heuristic algorithms are a popular way of training ANNs. A common way of achieving this is to have a predefined ANN structure, and then represent the weights of the ANN as a vector. Additionally, certain classes of Meta-heuristic algorithms, such as variations of Evolutionary Algorithms (EAs) [SM02, MH12, She11], are also capable of evolving ANN topologies.

Meta-heuristic algorithms are advantageous in that they are domain agnostic, meaning that they do not need to be modified for different task domains, as long as the representations of the candidate solutions are consistent, and the fitness function is tailored to the problem. It is for these reasons that we investigate these algorithms, more specifically *Neuro-Evolution* (the use of EAs to evolve ANNs), in the context controlling high-level crowd behaviour. We chose EAs in particular as they have already been applied to many similar problems, such as predator-prey scenarios [YM09] and strategy games [BM03], and have shown promising results.

3.2 Evolving Artificial Neural Networks

One popular approach to training ANNs is to use Evolutionary Algorithms (EAs) to optimize their structure and weights. EAs are a paradigm of population-based meta-heuristic algorithms inspired by biological evolution. In EAs, a population of candidate solutions is iteratively improved through various operators, the most common being: crossover; mutation; and selection. The primary functions of these three operators are as follows:

- Crossover combines selected individuals together in order to form offspring that contain a function of the genes from the parents. Figure 3.4 shows an example of a discrete crossover operator, where the offspring contains genes from both parents. It should be noted that certain classes of EAs, such as Evolutionary Strategies (ESs), typically do not use crossover.

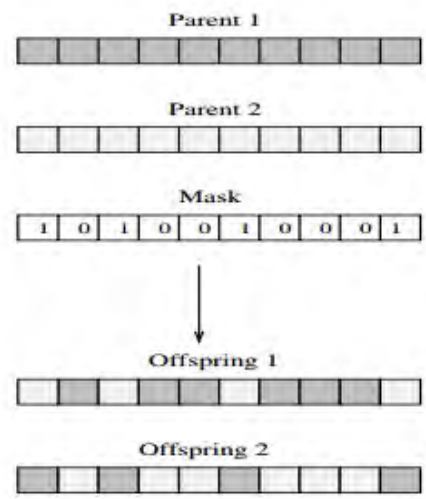

Figure 3.4: *An example of a crossover operator that creates a mask and uses it together with two parent chromosomes to generate two offspring with opposite genes (Source: [Eng07]).*

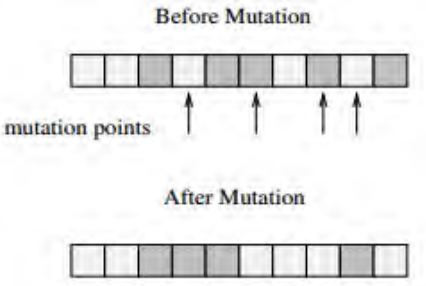

Figure 3.5: *A mutation operator that flips a random set of genes in a candidate solution (Source: [Eng07]).*

- Mutation (figure 3.5) adds noise to the genes of individuals, and is used increase genetic diversity in the population.

- Selection is used both to select the individuals for crossover, and to determine which individuals survive to the next generation. Selection is typically biased towards better performing individuals, which results in the population being evolved in the direction of these individuals.

There has been significant work on using EAs to train ANNs. This field has been termed Neuro-Evolution (NE). NE has achieved great success in training ANNs, particularly in the task domain of evolving agent controllers. Accordingly, the benchmark for NE algorithms is the *double pole balancing* problem, where the ANN is required to control a cart on a limited rail with two poles attached to it, and aim to move the cart in such a way so that the poles are prevented from falling. Double pole balancing exists as an extension to the single pole balancing problem, which is used as a benchmark for many fields, including engineering and reinforcement learning [SC66, MC68, And89], reason being that it is both intuitive and consists of many aspects of temporal credit assignment [GM99, Sut84].

NE algorithms can be divided into three main categories, based on how they evolve ANNs. *Single population fixed topology* approaches typically encode ANNs as vectors of weights, which can then be evolved as a n-dimensional real-space problem. *Cooperative Co-evolution* methods aim to separate the search space into smaller components and then evolve those components separately. *Topology and Weight Evolving Artificial Neural Network* algorithms aim to evolve both the topology and the weights of ANNs.

3.2.1 Single Population Fixed Topology

A straightforward method for evolving ANNs is to have a predefined network topology, with the chromosomes encoded as vectors of real numbers representing the weights within the ANN. These chromosomes are subsequently evolved and improved inside a single population. One of the earliest examples of these techniques comes from the work of Wieland [Wie91], who used a Genetic Algorithm to evolve ANN controllers for single and double pole balancing tasks. More recently, Gomez [GM03b] constructed an algorithm based on this approach for comparison purposes, which he named Conventional Neuro-evolution (CNE). However, it differs from Wieland's method in that it uses real numbers to encode the ANN as opposed to binary, and it uses burst mutation [GM03a] and rank-based selection [GD91].

A recent development in real-space EAs comes in the form of the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [HK04]. CMA-ES is used to optimize problems in a continuous domain. It achieves this by iteratively estimating a covariance matrix, mean, and step size, which define the multivariate distribution from which the population in the succeeding generation is sampled.

CMA-ES can be considered state-of-the-art in real-space EAs, and provides numerous benefits over its counterparts, including:

- It requires far less parameter tuning than many other EAs, as many of its parameters are self-adapting;

- It prevents premature convergences more effectively due to large population spread;
- It is a second order approach. Thus, for complex landscapes, the evolutionary path taken is much more direct than first order methods such as gradient descent;
- It does not utilize gradient information, and thus can be used on non-continuous and non-smooth domains; and
- It estimates the covariance matrix using information from past and current generations, thus allowing for far smaller population sizes compared to other EAs.

Due to the vast array of benefits that CMA-ES provides, its good results in real-space optimization, and the often complex dependencies between the weights of ANNs, it has also been applied to evolving the weights of ANNs [HMI09]. This approach is termed *CMA-NeuroES*. A more detailed explanation of this algorithm can be found in section 4.1.3.

3.2.2 Cooperative Co-evolution

One problem that EAs, along with various other meta-heuristic algorithms, suffer from in complex task domains is *two steps forward, one step back* [VdBE04]. This problem is characterized by the fact that while certain components of the candidate solution are moving closer to the optima, others can move away so long as the candidate solution provides an improvement over previous ones. This is caused by the fitness typically being evaluated only after a candidate has been fully constructed or updated.

One possible way of dealing with this is to evaluate the fitness of the candidate solutions after each component update [VdBE04], however, there are a few problems with this approach:

- It becomes very computationally expensive as the number of fitness evaluations increase with the number of components within the chromosome. This is a problem since fitness evaluations are already the bottleneck in population-based meta-heuristic algorithms;
- It assumes that the components are not correlated;
- In certain EAs, offspring are generated from sampling multivariate distributions. Since this process is atomic, the above method is not viable.

Potter and De Jong [PDJ94] propose a solution to this problem of *two steps forward, one step back*, termed Cooperative Co-evolution. Instead of having a single population of candidate solutions, they rather simultaneously evolve a set of sub-populations, where the individual members of each sub-population represent a portion of a candidate solution. Evolution is performed only within these sub-populations, with the exception being the fitness evaluation stage, where members from each sub-population are selected and combined with members from other sub-populations in order to form a candidate solution. Cooperative Co-evolution, however, assumes that there is no correlation between components. Thus, if correlated components end up in different sub-populations, poor performance results [VdBE04].

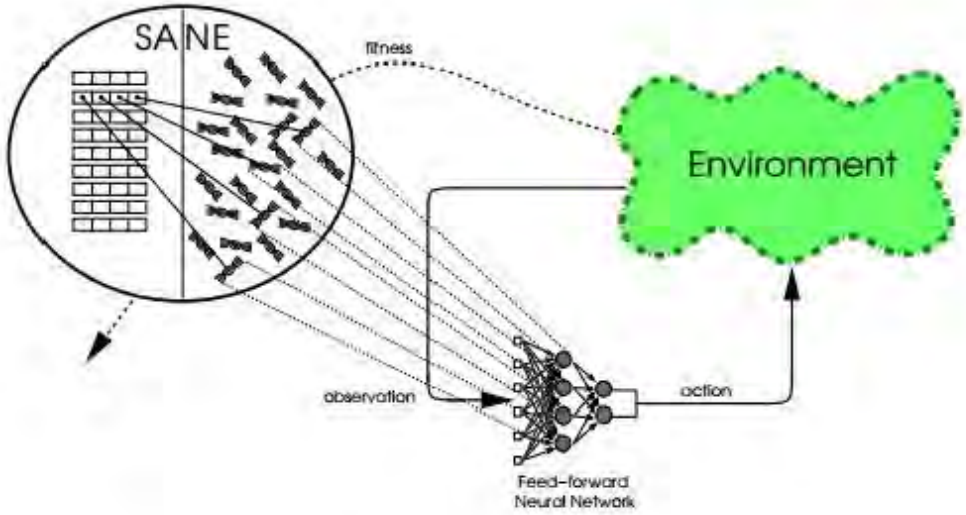

Figure 3.6: *SANE uses two sub-populations, one of blueprints and another of neurons. A candidate solution is formed by combining the neurons according to a selected blueprint, after which it is evaluated and the fitness shared between the participating neurons and the blueprint. (Source: [GM03b])*

Despite the drawbacks of Cooperative Co-evolution, it has been applied to ANNs with great success. The first attempt at this was called Symbiotic Adaptive Neuro-Evolution (SANE) [MM96]. In SANE (figure 3.6), two populations are evolved simultaneously, a population of neurons, and a population of ANN specifications that detail the structure of the ANN, with the user specifying the number of input and output nodes. An ANN is constructed for fitness evaluation by selecting neurons and a blueprint, this ANN is then evaluated in the task domain and the fitness is shared amongst all the participating neurons and the blueprint. This process is applied multiple times for each neuron and blueprint as the fitness evaluation process is noisy, since good neurons can be paired with poor neurons or blueprints. One interesting characteristic of SANE is that towards the end of evolution, the population of neurons tend to form clusters around good neurons as opposed to converging towards a single point [MM97], resulting in the specialization of the neurons and the avoidance of cases where a single good neuron dominates the population.

One problem with SANE, caused by the lack of explicit neuron specialization, is that it is poor at evolving Recurrent Neural Networks (RNNs), which are ANN structures with feedback loops [GM99]. Gomez [GM99] thus proposes Enforced Sub-Populations (ESP), an extension to SANE which allows for RNNs. In ESP (figure 3.7), a sub-population of neurons is created for each neuron in the ANN structure, thus allowing for each neuron to specialize. In order to evaluate the fitness of these neurons, one is selected from each sub-population and then used to construct the candidate solution, which in turn is evaluated in the task environment. The fitness is then shared across all participating neurons. Like SANE, the neurons in ESP also need to be evaluated multiple times in order to combat noisy fitness evaluation. The main advantage of ESP over SANE is that, since each

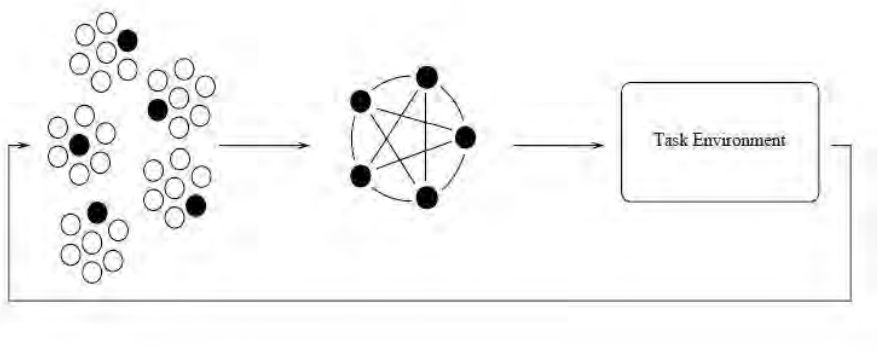

Figure 3.7: *ESP separates the neurons into multi sub-populations, with the candidate solution being constructed by selecting a neuron from each sub-population (Source: [GM99]).*

neuron has its own sub-population, one can specialize the neurons to allow for a variety of non-fully connected and recurrent structures.

In certain problems, it is required that the candidate solution consists of multiple ANNs. An extension of ESP, termed Multi-Agent ESP [YM09], allows for this. Multi-Agent ESP is identical to ESP except that each sub-population ahas the information of which ANN in the candidate solution it belongs to, in addition to its position in the ANN. Multi-Agent ESP has been tested in the predator-prey problem domain, with the agents evolved achieving interesting behaviours [YM09], such as role specialization and communicating via stigmergy.

A problem with both SANE and ESP is premature convergence. A more recent algorithm, Cooperative Synapse Neuro-Evolution (CoSyNE) [GSM06], deals with this by evolving ANNs cooperatively at an individual weight level by creating a sub-population for each weight. These weights can then be encoded in a matrix where the rows represent the sub-populations and the columns individuals within each sub-population. In order to evaluate the fitness of these individuals, an ANN is constructed for each column in the matrix, and then evaluated in the task environment. Recombination is then performed on the best quarter of ANNs, with new weights replacing the worst weights within each row. In order to maintain genetic diversity within the sub-populations, the rows are then permuted in order to construct different networks in the succeeding generation. CoSyNE can be considered state of the art in Cooperative Neuro-Evolution, and has achieved excellent results in the double pole balancing problem, outperforming most of its competitors [GSM06]. Despite the high performance of CoSyNE on double pole-balancing, there is a lack of literature on it being applied to other domains. Thus, it is unclear on how well it generalizes to other problems.

3.2.3 Topology and Weight Evolving Artificial Neural Networks

In addition to the weights, topology also plays a part in determining the functionality of an ANN, as it determines the information flow between neurons. In algorithms where only the weights are evolved, the topology has to either be pre-determined or pruned afterwards. This, unfortunately, requires either trial and error for pre-determining network structure, or significantly more processing time when pruning the networks post training. To overcome this, there are a subset of NE algorithms that aim to evolve both

the weights and the topology of ANNs, termed Topology and Weight Evolving Artificial Neural Networks (TWEANNs). Their main benefit is that they allow one to save time that would otherwise be wasted tuning the network topology [GWP96].

One of the simplest TWEANNs is Dasgupta and Mcgregor's Structured Genetic Algorithm (sGA) [DM92], which uses a connectivity matrix, encoded as a bit-string, to represent the connections between the nodes in the ANN, with 1 being an active connection, and 0 being inactive. This connectivity matrix, along with the connection weights, is then evolved in a standard GA. One benefit of the connectivity matrix representation is that it can be evolved without any modifications to the conventional Genetic Algorithm, since it can be represented as a bit-string. However, this representation is quite memory intensive($O(n^2)$), requires a fixed maximum number of nodes because all the bit-strings must be the same length, and is prone to poor resultant structures as evolving bit-strings makes it difficult to obtain intelligible graph designs. Another notable drawback is that the weights have to be evolved for all the connections regardless of whether they are active or not, typically leading to a very high dimensional search space even on medium sized ANNs.

Parallel Distributed Genetic Programming (PDGP) [PP98] is a TWEANN that uses a dual-representation to evolve ANNs. In PDGP, both a grid and a linear representation are used. This makes it possible to incorporate different types of operators for different representations. For example, sub-graph swapping and topological mutations are performed on the grid, whereas weight crossover and mutation operations are performed on the linear representation. Compared to sGA, the more explicit graph structure in PDGP allows for more intelligible evolution of the ANN topologies, as it is possible to design graph specific operators. However, it still suffers from many of the same problems of sGA, such as a set maximum node size and the memory expense.

Neuro-Evolution of Augmenting Topologies (NEAT) [SM02] is a TWEANN that has gained significant popularity since its inception. NEAT represents an ANN's chromosome as a series of connections, with each connection having a from and to node, and can be either enabled or disabled. Each connection is also provided with an *innovation number*, which allows for easy comparison of differing structures without the use of expensive topological analysis algorithms.

In order to evolve these structures, the population is divided up into disjoint sets based on similarities between the ANN structures, with evolution between genes occurring within these sets (also termed *species*). This process is termed *Speciation*. Typically, adding a connection to the ANN structure will result in an initial dip in the ANNs task performance, until the connection weights can be optimized. Speciation deals with this problem by only allowing chromosomes to compete within their own species, allowing them to optimize their weights first before competing with the entire population. The speciation process in NEAT also uses explicit fitness sharing, where the fitness of the ANNs are normalized according to the population size of that species. This prevents a good species from dominating the entire population, which may result in premature convergence. Speciation also deals with the *Competing Conventions* (figure 3.8) problem of TWEANNs. This is when ANNs with similar functionalities have vastly different structures, with a crossover between the two resulting in damaged offspring. This problem is remedied by speciation as a chromosome can only reproduce with another if it is in the same species.

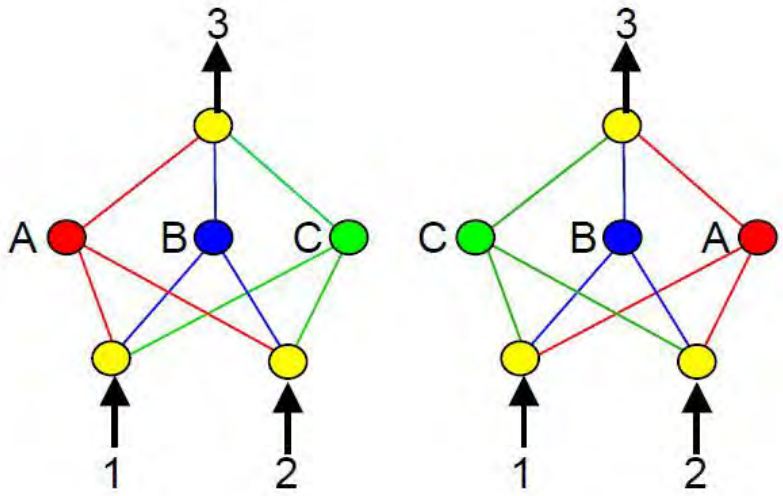

Figure 3.8: *An example of the competing conventions problem (Source: [SM02]).*

A problem with many TWEANNs is that since the population of topologies is randomly initialized, they often have many poor initial connections, which are then maintained as long as the ANNs perform well in the given task. NEAT deals with this by initializing genes with no hidden nodes in the initial population. As evolution progresses, the ANN structure is extended through mutation, with good extensions surviving to succeeding generations. This approach allows for a minimalistic evolutionary process, where structural connections have proved themselves throughout the evolutionary process. This minimalistic approach to evolving structures is termed *Complexification*.

3.3 Automated Controller Design

Automated controller design, a field concerned with the automated creation and tuning of robot and simulated agent controllers, has been a popular area of study in Artificial Intelligence in the last few decades. There has been a variety of methods used in this context, including optimizing Fuzzy Controllers [Bab98], Evolutionary Algorithms [MC96, QSMH03], Reinforcement Learning [KLM96], Swarm Intelligence [BK11, JLLW08], and ANNs [SBM05, PSY88].

NE has also been prominent in this field, and has been used in a variety of agent control tasks, such as video games [MM95, TL05, FHHQ04], pole balancing [SM02, GM03b], and robotics [FMF+94, Bro89]. This section is aimed at exploring the various techniques, issues, and challenges for automated controller design, primarily in the context of NE.

3.3.1 Controlling Agents

A popular application area of NE is that of evolving agent controllers, with perhaps the most well known problem being pole balancing. This control can be achieved by representing the various performable actions as output nodes in the ANN, with the output values of these nodes being used to determine the behaviors of the agent.

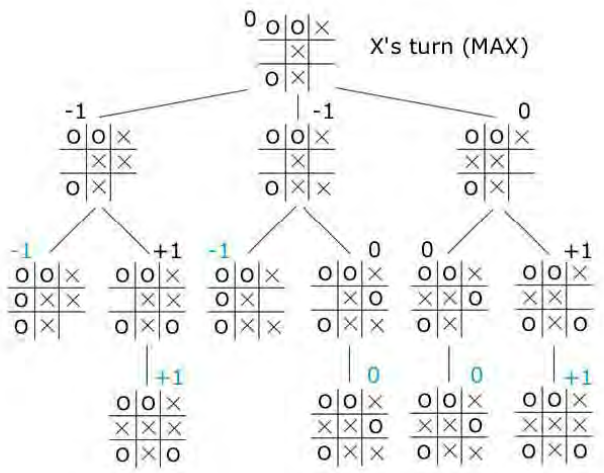

Figure 3.9: *A Game Tree for Tic-Tac-Toe. Each branch in this tree represents a move, whereas each node represents the future state corresponding to the taken move (Source: [Gam]).*

One way to interpret these output values is to view them as reward values, with the agent's action corresponding to the node with the highest reward chosen as the action to be performed for that iteration. An example of this approach would be in the work of Moriarty and Miikkulainen [MM95], where outputs as rewards are used when evolving an agent to play *Othello*. The ANN that they used provided 64 outputs, with each output corresponding to a position on the board, and the activation value interpreted as how strongly the ANN recommended moving to that position. Other applications of this output approach involves teaching ANNs to make high-level decisions in the Real-Time Strategy (RTS) game *Wargus* [TT12], and teaching ANNs to play Go [RMM98].

Choosing a specific action based on the output values of the output nodes has been applied with some success, especially in certain game playing environments where only 1 action is possible at every iteration. However, it is limited because it cannot determine how strongly an action is performed (for example turn by how many degrees), and cannot be used to perform multiple actions per iteration. Additionally, this approach does not scale well as various discrete actions, such as turning, can be aggregated into a single output node since negative values can be used to signify opposite actions.

Another method used in game environments is to replace the heuristic in *game trees* (figure 3.10) with an ANN. A game tree details the various states of the game as nodes, with possible moves as edges (figure 3.10). Chellapilla and Fogel [CF99] use this approach to train an ANN to play checkers by using a vector of 32 inputs to represent the state of the cells of the board, and one output node to determine the desirability of a state. After evaluating the various future states with the ANN, the system then traverses the edge that leads to the state that yields the highest output. This approach has proven popular with board games, and has been used in Chess [FHHQ04], Othello [MM94], and Go [LM01].

ANN driven heuristics work well with certain games, such as board games, as it scales well because of its single output. However, it is only really viable if there are only a limited number of possible moves at each iteration, as the game tree would otherwise become

too expensive to construct and traverse without proper pruning strategies. Another problem with this approach is that it does not work well with actions that can have variable intensities (for example, kicking a ball at various strengths) as one would need to discretise these intensities, leading to a very large branching factor for the game tree.

A popular method of using ANNs to control agents is to have the output values of the output nodes determine the extent of which an agent should perform an action. This approach is advantageous as it allows an agent to perform multiple actions on every iteration, and to differing degrees. One example [PP07] uses this approach when training an ANN to control agents within the 2D space combat game *XPilot*. The ANN is responsible for determining the turning rate, amount of thrust, and whether or not to shoot. Unlike thrust or turning, shooting is a binary action in XPilot. Thus, a threshold on the corresponding output node is used to convert the value to binary. Additional applications of this approach include teaching a bicycle agent to perform tricks [TGLT14], teaching car agents to race [TL05, TL06, TBL$^+$07, CLL09], teaching agents to play shooting games [SBM05, PB09], maze solving [LS08], and pole balancing [SM02, GM03b, GM99, MM96].

3.3.2 Controller inputs

In addition to the design of ANN outputs, modelling their inputs is also important. A simple way to model these inputs would be to use the global environment information, an approach that has been popular when training ANNs to play board games [MM95, CF99, Fog93].

One problem with modelling the ANN inputs with the global environment state is that many inputs are required for complex environments. This leads to a higher search space dimensionality, resulting in slower if not infeasible training times. Additionally, in some situations, such as First Person Shooter (FPS) video games, it might not be desirable to provide all the information as this may lead to the AI making over informed decisions, resulting in a possible break in immersion.

An alternative method of modeling the inputs would be to instead provide the agent's local view of the environment. This local input can be in a variety of forms, such as ray sensors [TL05, TL06, TBL$^+$07, CLL09, SBM05], cone sensors [SBM05, CLL09], relative angles, velocities, positions [PP07, TL05, TL06, TBL$^+$07], additional third person inputs such as damage taken [CLL09] and distance to goal [WGN14], and raw inputs such as the render information [PB09, FKMS04] and grids representing the environment directly surrounding the agent [TKKS09].

Local inputs are beneficial when the environment is too complex to fully represent as inputs. Additionally, they allow for the reduction in the amount of noisy information sent to the ANN, as well as allow for more realistic decisions in scenarios where limited information is provided. Despite this, the modelling of these inputs requires great care as the given inputs can bias the evolutionary process, which determines what types of strategies are discovered [SR14].

In addition to manually defining the sensory inputs of agents, there has also been promising work performed on co-evolving both an agent's body and controller [BBZ05, Sim94a, Sim94b]. These techniques show that it is possible to also evolve various aspects

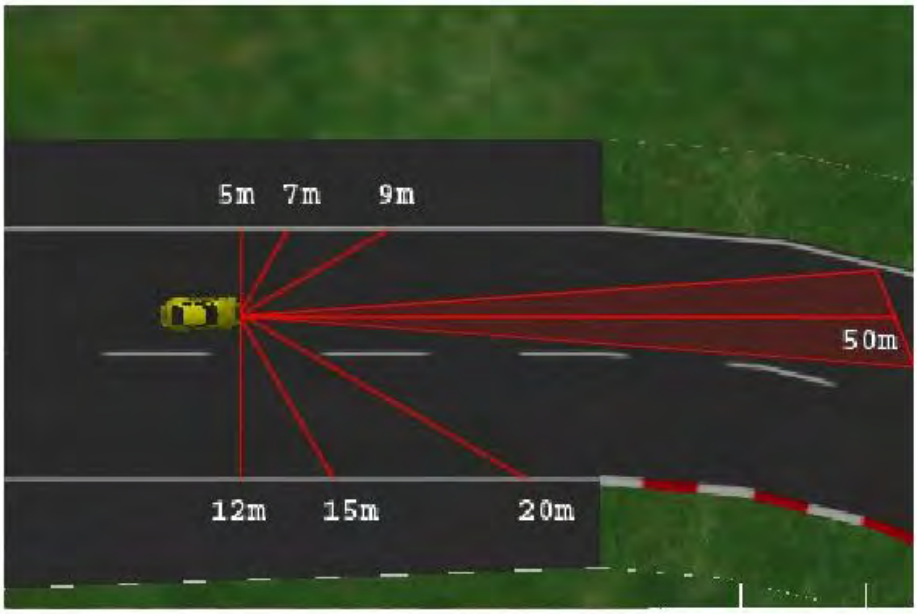

Figure 3.10: *Example of using ray and cone sensors to simulate a car agent's vision in the car racing simulator TORCS (Source: [CLL09]).*

of an agent's sensory inputs (for example vision resolution frustum size). However, these techniques have the disadvantage of further increasing the problem search-space.

3.3.3 Evolving for multiple objectives

The performance of candidate solutions in NE is typically measured in the form of objectives. For straightforward scenarios, such as pole balancing and maze solving, a single objective is sufficient. However, given more complex tasks, there are often multiple conflicting requirements (for example training FPS agents to deal maximum damage while receiving minimum damage), giving rise to the need for multiple objectives. As most learning methods learn only according to a single objective, it becomes necessary to introduce new methods that allow for multiple objective training [SM08]. These techniques are useful for controlling crowd simulations, as there are often multiple control requirements for a single simulation scenario.

Treating multiple objectives as a single objective

One way of achieving the evolution of multiple objectives is to represent the fitness of candidate solutions as the weighted sum of these objectives [SM08, SBM05]. Since this method combines the multiple objectives into a single scalar value, a major benefit is that this approach can be used with most other already existing training algorithms. Additionally, this approach is more computationally efficient than other multi-objective methods [Coe99]. A major drawback is that the performance of the optimization is determined largely by the objectives' weights, which are difficult to determine without sufficient prior information on the problem [Coe99]. Other approaches that aim to try reduce the multiple objectives to a single objective include Goal Programming [CC57], Goal Attainment [CL94], and ϵ-Constraint [RER94].

Pareto-based multi-objective optimization

A different strategy for optimizing multiple objectives is to instead use pareto-based approaches, which aim to generate a set of non-dominated pareto-optimal solutions from which a final decision can be made [Coe99]. Examples of these techniques include the Multi-Objective Genetic Algorithm [FF$^+$93], Non-Dominated Sorting Genetic Algorithm (NSGA) [SD94], and Niched Pareto Genetic Algorithm [rHNG93].

Schrum and Miikkulainen [SM08] studied Pareto-based multi-objective optimization when training agents in a battle domain where they have to deal maximum damage to a monster while in turn receiving minimal damage. In their work, they compared the weighted sum approach with the pareto-based genetic algorithm *Non-Dominated Sorting Genetic Algorithm-II* (NSGA-II) [DPAM02] across three different scenarios. They found that NSGA-II outperformed the weighted sum approach on two of the three scenarios, and noted that with NSGA-II, the agents were capable of learning more strategies compared to the weighted sum approach.

An interesting application of multi-objective optimization is introduced in the work of Agapitos et al. [ATL$^+$08]. They use NSGA-II to generate a diverse array of opponents in a car racing game, by providing a series of semi-conflicting fitness measures. This is possible because pareto-based techniques generate a population of solutions on the pareto front. Thus, allowing for the analysis and selection of differing solutions in order to ensure that the agents perform differently.

3.3.4 Multi-Agent Systems

Many applications, such as crowd simulations, require the simulation of many agents in a shared environment. These simulations, termed Multi-Agent Systems (MASs), differ from their single-agent counterpart in that one has to take into account factors such as cooperation or competition between the agents, agent communication, and role specialization. These factors can be divided up into two axes [SV00], namely *communicative vs non-communicative* and *homogeneous vs heterogeneous*.

3.3.5 Homogeneous versus Heterogeneous teams

One method of designing an MAS is to have separate controllers for each agent [HRBvG95, YM09, Bal98]. This approach, termed *Heterogeneous teams* [SV00, BM03], has the benefit that agents within a team are able to have specific roles [SV00]. This allows agents to have simple controllers that can cooperate with other agents in order to accomplish complex tasks.

A problem with role specialization, however, is that the failure of one component may lead to the failure of the entire team [BM03]. This has, however, been addressed by the works of Prasad et al. [PLL98], where they allow agents to switch roles dynamically in the case of failure. Another issue with heterogeneous teams occurs when using machine learning techniques to train controllers for large numbers of agents, as the large number of controllers needed to be trained together results in large search space dimensionality, which becomes infeasible to train in a realistic amount of time.

A different approach to simulating MASs is *Homogeneous teams*, where the agent controllers are identical. This has the benefit that only one controller needs to be designed,

which works well in machine learning as it drastically reduces the search space. An issue with homogeneous teams is behavioural diversity, as agents have identical controllers. Sen et al. [SAR98] deal with this in a resource gathering use-case by providing the agents with local inputs, as opposed to global. In this use-case, agents are required to gather various resources, with under-used resources being prioritized higher. If the agents had identical controllers and global inputs, they would all move towards the same resource node. With local inputs, however, their behaviour would deviate as they would move towards the best local resource node, rather than the global one.

Both homogeneous and heterogeneous teams have been investigated in NE. An example of using heterogeneous teams in NE comes from the work of Yong and Miikulainen [YM09], where they train predator agents in the *Predator-Prey* domain. In this domain, predator agents are tasked to catch a prey agent. The prey agent is given a velocity equal to or greater than that of the predator agents, thus cooperation between the predators is required in order to catch the prey. Yong and Miikulainen [YM09] found that the agent controllers specialize, with the predators learning roles such as chasing and blocking.

Homogeneous teams have also been studied in the context of NE, with one example being the work of Bryant and Miikkulainen [BM03], where they trained ANNs to play the *Legion-I* game. Legion-I is a tile based game where the player (or in this case ANN controllers) has to control legion agents in order to prevent barbarians from pillaging the land. In the game, there are two types of tiles, garrisons and farmland. If a barbarian pillages a farmland tile, the player loses one point, whereas if a barbarian pillages a garrison, the player loses 100 points. In the game, there are more legions than garrisons, thus, the ideal strategy would be to have two types of legions, one to garrison and another to search and remove barbarians. In their approach, they assigned an identical ANN controller to each agent, with the agents providing local inputs to this controller in order to determine their desired behaviour. It was found that not only did the legions deviate in their behaviour, they were also able to adopt different roles based on their local context, showing that agents are capable of adopting significantly different behaviours based only on their local perception of the environment.

A more in depth study on team composition in NE is performed by Waibel et al. [WKF09]. In their work, they tested the different approaches on three different simulations, and found that the performance of the type of team used depends largely on the amount of cooperation required between the agents. They found that in scenarios where cooperation is not required, heterogeneous teams perform better, whereas in scenarios where cooperation is important, homogeneous teams perform better.

3.3.6 Communicative versus Non-communicative agents

As agents are often required to cooperate in MASs, it is important to model their communications. This can be achieved by either having the agents explicitly communicate, or by relying on *stigmergy*, which is communication through observing the effects that other agents have on the environment.

Although explicit agent communication is intuitive to understand, care needs to be taken with regards to what content needs to be sent. One type of content is agent sensor data, where an agent communicates its perception of the environment. An example of this is in the work of Tan [Tan93], where he allows predator agents in the predator-prey

domain to communicate their local environment perceptions. Communicating sensory inputs is shown to be useful for tasks where agents have a limited perception of the environment, but require more information to accomplish the task at hand, as it allows agents to enhance their sensing capabilities via mutual scouting [Tan93]. It was noted, however, that superfluous sensory information leads to interference with the learning process. In addition to sensory information, agents are also able to directly communicate their internal state and goals. Stone and Veloso [SVR99] investigate this method in the robot soccer domain, finding it to be very effective in enabling the agents to accomplish their goals.

Stigmergy is another type of communication, where conveying messages across agents is achieved through environmental interactions. Beckers et al. [BHD94] separates stigmergy into two different types. The first type is called *active stigmergy*, where an agent alters the environment in order to affect other agents' senses. Goldman and Rosenchein [GR94] use this in the *Tileworld* domain, where agents have to move around a world and push tiles into holes. They modelled active stigmergy by having agents move tiles, that they encounter but are not working on, away from obstacles in order to facilitate easier problem solving for other agents which interact with the tile in the future.

The second type of stigmergy is termed *passive stigmergy*, where an agent's actions that alter the environment affects the future actions of other agents. Holland [Hol96] use this type of stigmergy when simulating a colony of robotic ants. The ants employ passive stigmergy in a corpse gathering task, where the density of corpses in an area determines their probability of picking up and dropping corpses. Passive stigmergy helps cooperation in this task as eventually the corpses will be gathered into one heap.

Inter-agent communication has also been recently studied in NE. One of the better known instances is the work of Yong and Miikkulainen [YM09], where they use Multi-Agent ESP to control predator agents in the predator-prey simulation. In their experiments, they compared the performance of a group of agents with explicit sensory communication with a group with stigmergic communication. They found that the group of agents that had no explicit communications performed just as well, and more robustly, than the group of agents that did, showing that stigmergy performs well in this task.

A later work by Rajagopalan et al.[RRM+11] investigates this issue further, and notes that although stigmergy performs well in simple tasks, explicit communication performs better in more complex tasks. Additionally, the authors also note that shared fitness between the agents strongly promotes cooperation.

3.4 Summary

This chapter introduces some theoretical concepts of ANNs, as well as methods of training them. Additionally, it provides an in depth background on NE and its applications.

The NE techniques introduced were separated into three categories: single population fixed topology, cooperative co-evolution, and TWEANNs. Although the algorithms from these various categories have been benchmarked in the double pole balancing domain, there is little indication of their performance in the context of controlling crowd simulations. For this reason, we aim to compare well established algorithms from each category in order to determine what type of algorithm is preferred in this domain.

Our specific algorithms of choice for each category are CMA-ES for single population weights only methods, NEAT for TWEANNs, and ESP for cooperative co-evolutionary methods. ESP was chosen over CoSyNE, as although CoSyNE has performed very well in the double pole-balancing domain, there has been a lack of literature of it being applied to other domains. ESP, on the other hand, has already been studied and shown to perform well in MASs [YM09]. In addition to these three algorithms, we will also perform tests for CNE due to its simplicity. This will allow us to evaluate if more complex algorithms are actually necessary in this task domain.

In addition to training algorithms, some of the background work performed in MASs provides insight on how to model and design our simulation scenarios and agents. One such insight is that of communication modelling. Explicit communications is infeasible in crowd simulations due to the large numbers of agents, and active stigmergy requires us to model how an agent behaves as it needs to deliberately alter the environment, which is not possible as we are aiming to evolve behaviours rather than define them. Passive stigmergy, on the other hand, appears to be a promising approach as it allows the agents in our simulations to communicate in a scalable and evolve-able manner.

Another issue that needs to be addressed is whether to use homogeneous or heterogeneous teams. It is important to have agents with diverse behaviours when simulating crowds for films, as agents with overly similar behaviours leads to a lack of believability in the scene. Although it has been shown that both homogeneous and heterogeneous teams are capable of generating these diverse behaviours, it remains an open question as to which performs better, both in achieving the desired behaviours, and in generating the more *believably diverse* behaviours. Therefore, we will also aim to investigate what team compositions perform well in the context of controlling high-level crowd behaviours.

Chapter 4

Controlling Crowds with Neuro-Evolution

The main aim of this thesis is to use Neuro-Evolution (NE) to control crowds for the potential use in films. In order to achieve this, we have separated our system into three sub-systems (Figure 4.1): *NE*, responsible for evolving the ANNs; *Simulation*, a distributed sub-system responsible for running the crowd simulation scenarios and evaluating the fitness of the ANNs; and *Rendering*, responsible for visualizing the crowd simulations. This chapter will aim to detail the design and workings of the NE sub-system. The other two sub-systems are further elaborated in the succeeding chapter.

4.1 NE Algorithms

The NE sub-system is responsible for evolving the ANN controllers so that the crowd behaviour adheres to the user-defined requirements. This is achieved by evolving a population of ANNs with one of the four implemented NE algorithms, using the simulation sub-system for fitness evaluations. The way we store and define the ANNs is through XML files (appendix A).

The four NE algorithms that we have implemented for this sub-system are Conventional Neuro-Evolution (CNE) (section 4.1.1), Enforced Sub-populations (ESP) (section 4.1.2), Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) (section 4.1.3), and Neuro-Evolution of Augmenting Topologies (NEAT) (section 4.1.4). These algorithms were implemented to represent the different categories of direct-encoding NE algorithms. CNE was implemented as it is arguably the simplest algorithm for evolving ANNs, and gives us an idea if more complex NE algorithms are actually necessary for solving our problem. This section further elaborates on the brief explanations of these algorithms, found in section 3.2.

4.1.1 Conventional Neuro-Evolution (CNE)

An easy method of evolving ANNs is by assigning them a fixed topology, and then evolving the weights of the networks as vectors of real numbers (figure 4.2). This approach was first introduced by Wieland [Wie91], and is termed *Conventional Neuro-Evolution* (CNE), as

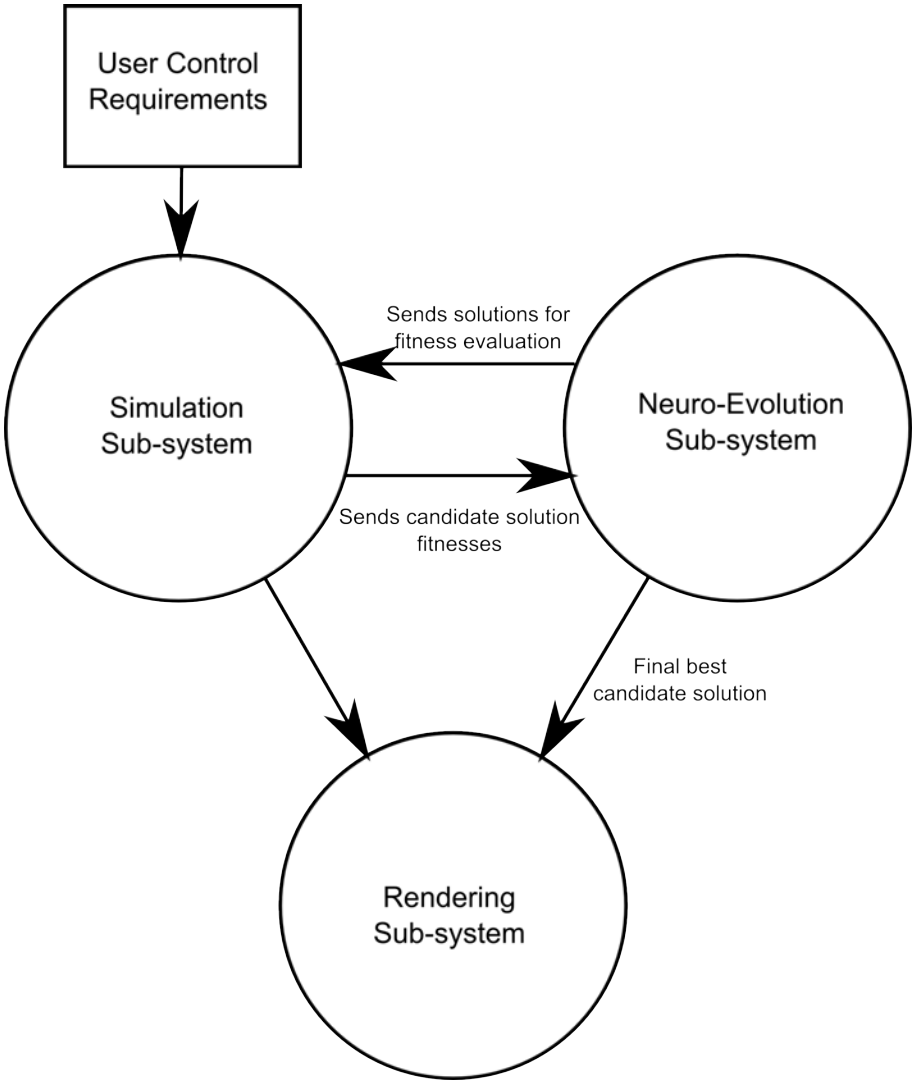

Figure 4.1: *Architecture of our system.*

Algorithm 1: CNE

Initialize chromosome population P;
Evaluate(P);
while *Termination criteria not met* **do**
    Select parents S from P using some selection operator;
    Produce offspring O from S using some crossover operator;
    Mutate O using some mutation operator;
    Evaluate(O);
    Select chromosomes P_{new} from P and O using some selection strategy;
    set P to P_{new};
end
return P_{best}

was later coined by Gomez [GM03b], who used the algorithm as a basis for comparisons in his experiments. CNE is detailed in algorithm 1.

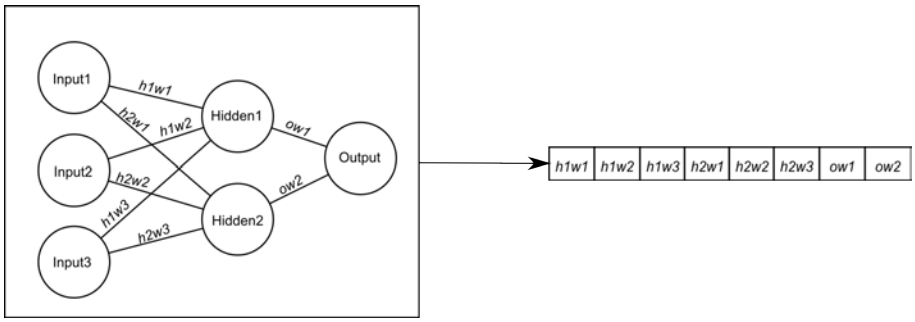

Figure 4.2: *CNE process of converting the weights in an ANN to a vector of real values.*

Since the ANNs are represented as a vector of real numbers, one is able to select from a large pool of existing GA operators. The operators implemented in this system are One-Point Crossover, Two-point Crossover, Uniform Crossover, Simulated Binary Crossover (SBX), Arithmetic Crossover, Blend Crossover (BLX-α), Heuristic Crossover, Uni-modal Normal Distribution Crossover (UNDX), Parent Centric Crossover (PCX), Laplace Crossover, Uniform Mutation, Gaussian Mutation, Cauchy-Lorentz Mutation, Linear Rank-based Selection, Non-Linear Rank-based Selection, Tournament Selection, and Boltzmann Selection. These operators are elaborated in sections 4.2 - 4.4.

4.1.2 Enforced Sub-Populations (ESP)

Enforced Sub-populations (ESP) is a cooperative co-evolution approach to NE. Much like CNE, ESP also evolves ANNs with fixed topologies, with the weights being represented as vectors of real numbers. However, unlike CNE, ESP evolves ANNs at a neuron level, having a separate sub-population for each neuron position (figure 4.3). Although in Gomez's original work [GM03b], only hidden neurons were evolved with output neurons being statically defined, we decided to also evolve the output neurons as initial experimentation showed that doing so is beneficial to the speed of evolution. The pseudo-code for ESP appear in Algorithm 2.

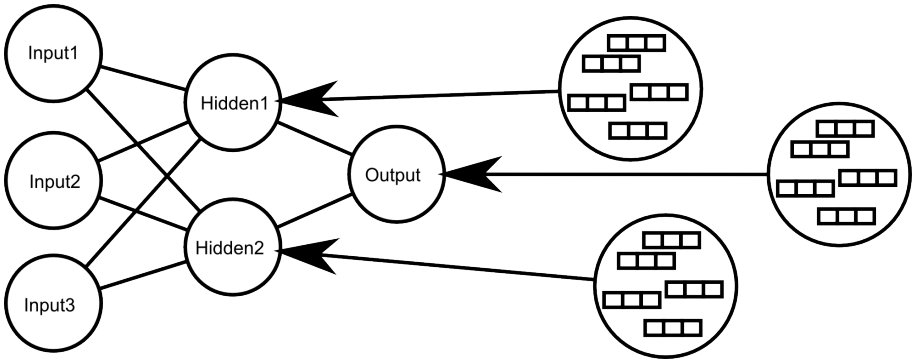

Figure 4.3: *In ESP, a sub-population of neurons is evolved for every neuron position in the ANN structure.*

Algorithm 2: ESP

```
Set n to amount of neurons in ANN structure;
Initialize n sub-populations P_1..P_n;
Evaluate(P_1..P_n);
while Termination criteria not met do
    foreach P_i do
        if P_i stagnated then
            BurstMutate(P_i);
        else
            Select parents S_i from P_i using some selection operator;
            Produce offspring O_i from S_i using some crossover operator;
            Mutate O_i using some mutation operator;
        end
    end
    Evaluate(O_1..O_n);
    foreach O_i do
        set P_i to O_i;
    end
end
return P_ibest
```

In order to evaluate the fitness of each neuron, the following process is executed: For a given sub-population, each neuron is systematically selected and evaluated in company with neurons randomly selected from each of the other sub-populations. Thus, the evaluation of each neuron is in the context of a complete ANN. The fitness achieved by the constructed ANN is then shared amongst all the participating neurons.

One problem with this approach is that a good neuron can get paired with poor neurons, resulting in it achieving a poor fitness rating. The way ESP combats this is by evaluating each neuron multiple times, and then averaging the obtained fitnesses. This decreases the amount of noise in the fitness evaluation process.

Another problem with ESP reported by Gomez [GM03b] is that, in the case of difficult problems, the sub-populations often converge prematurely to a sub-optimal area in the search space. In order to combat this, burst mutation [GM03b] is used on a sub-population whenever that population stagnates. This re-initializes the entire population in a distribution (typically Cauchy) around the current best chromosome (figure 4.4). We detect stagnation in our ESP implementation by checking if a sub-population's best chromosome has not improved for n generations.

4.1.3 Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)

Like CNE, CMA-ES is a single population approach that evolves the weights of an ANN by representing them as a vector of real numbers. However, CMA-ES does not make use any of the operators described in sections 4.2 - 4.4. Instead, it estimates a covariance

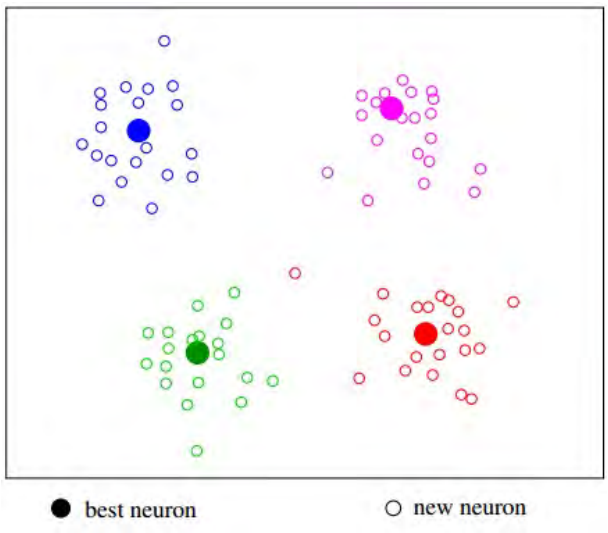

Figure 4.4: *Burst mutation re-initializes the entire population in a distribution centred around the best chromosome in the current population (Source: [GM03b]).*

Algorithm 3: CMA-ES

Initialize $C = I$, $m = \theta_{init}$, $\sigma = \alpha_{init}$, $p_\sigma = p_c = 0$;
while *Termination criteria not met* **do**
    Generate population P by sampling a multi-variate Gaussian distribution with m as mean, σ as step-size, and C as the covariance matrix;
    Evaluate(P);
    Update m by computing weighted average of P;
    Update the step-size evolutionary path p_σ;
    Update step-size σ using p_σ;
    Update covariance matrix evolutionary path p_c;
    Update covariance matrix C using p_c;
end
return P_{best}

matrix, a step-size, and a mean, which are then used to define a Gaussian distribution from which the next generation is sampled. The covariance matrix and step-size are iteratively updated every generation so that the likelihood of sampling better candidate solutions increases. This is achieved by biasing the covariance matrix estimation towards better performing candidate solutions from the previous generations, and adapting the step-size so that the distribution scales effectively. Algorithm 3 provides a high level overview of CMA-ES, and the rest of this section will aim to elaborate on the specifics.

Generating new offspring

Offspring in CMA-ES are created by sampling a multi-variate Gaussian distribution [Mor90]. This is a Gaussian distribution defined over n dimensions, and can be visualized as an ellipsoid in hyper-space (figure 4.5). Equation 4.1 shows how to generate offspring, where $N(0, Y)$ is a Gaussian distribution with a mean of 0 and covariance ma-

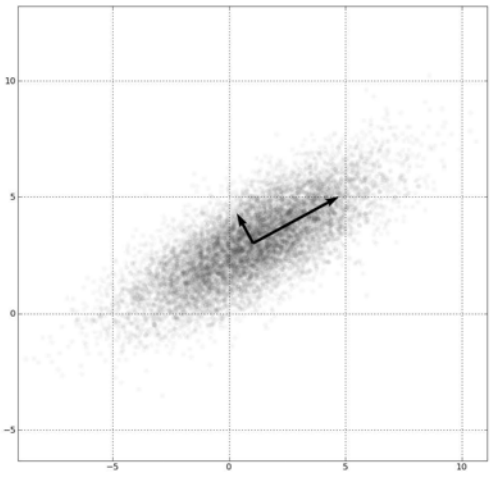

Figure 4.5: *A bi-variate Gaussian distribution (Source: [MVG]).*

trix Y, σ is the step size defining the scale of the distribution, and m is the mean of the distribution. Note that the mean here is not multiplied with the step size, as it would shift the distribution.

$$m + \sigma N(0, C) \tag{4.1}$$

In our implementation, we sample $N(0, C)$ by performing Eigen decomposition on the covariance matrix, and then multiplying the square-roots of the Eigen values obtained with values sampled from the uni-variate distribution $N(0, 1)$ [Han05]. Although, Cholesky decomposition can also achieve this by multiplying the lower triangular matrix obtained with values sampled from $N(0, 1)$ [WWW65], Eigen decomposition is preferred as it is also needed to update the step-size evolutionary path p_σ later in the algorithm.

Updating the mean

Updating the mean vector m allows the shifting of the distribution to more promising areas of the search space. In order to achieve this, a weighted average of the best μ chromosomes is calculated. These weights are determined by the fitness rankings of the current chromosomes, and are increased logarithmically (equation 4.2), with chromosomes that have a better fitness receiving a higher weighting. This results in biasing the mean towards more fit individuals, with the aim of moving the distribution towards better areas of the search space. The effect of using a logarithmic scale as opposed to a higher order one is that the algorithm ends up exploring more options, as many chromosomes will have similar weightings. Equation 4.2 shows how the weights are normalized and assigned and how the mean is computed, assuming the chromosomes are sorted from best to worst.

$$
\begin{aligned}
w_i &= \frac{w_i^{'}}{\sum_{j=1}^{\mu} w_j^{'}}, \\
m &= \sum_{i=1}^{\mu} w_i x_i, \\
w_i^{'} &= ln(\mu^{'} + 0.5) - ln(i), \\
\mu^{'} &= \frac{\lambda}{2}, \\
\mu &= \lfloor \mu^{'} \rfloor, \\
\lambda &= 4 + \lfloor 3ln(n) \rfloor, \\
i &= 1..\mu
\end{aligned}
\tag{4.2}
$$

In these equations, λ specifies total population size, μ the number of chromosomes selected, x_i the ith best chromosome, w_i the weight corresponding to x_i, m the weighted mean, and n the problem dimensionality.

Updating the Covariance Matrix

In addition to shifting the distribution, changing its shape also has an effect on how well it covers promising areas of the search space. This is achieved by changing the covariance matrix C defining the distribution.

The standard method of estimating a covariance matrix from a normally distributed population is provided by equation 4.3, where λ is the population size and x_i is the i^{th} individual. However, this only estimates an unbiased distribution. Thus, given a sufficiently large sample size, the distribution will not change over the generations.

$$
C = \frac{1}{\lambda} \sum_{i=1} \lambda (x_i - m)(x_i - m)^T \tag{4.3}
$$

In order to update the covariance matrix so that it covers the better performing regions, CMA-ES uses Rank-μ update (equation 4.4). This method combines the best μ performing chromosomes in a weighted sum manner (equation 4.2), resulting in the distribution for the next generation g being biased towards the better performing chromosomes from the current generation $g - 1$.

$$
C^g = (1 - c_\mu) C^{g-1} + \sum_{i=1}^{\mu} w_i ((x_i - m)(x_i - m)^T) \tag{4.4}
$$

$$
c_\mu = min(1 - c_1, 2 \frac{\mu_{eff} - 2 + 1/\mu_{eff}}{(n+2)^2 + \mu_{eff}}) \tag{4.5}
$$

$$
\mu_{eff} = (\sum_{i=1}^{\mu} w_i^2)^{-1} \tag{4.6}
$$

To allow for faster evolution, λ is typically a small value, as it results in fewer fitness evaluations per generation. However, this is problematic as a sufficiently large sample size is required in order to achieve a reliable estimation for the covariance matrix. In order to alleviate this problem, a learning rate c_μ (equation 4.5) is used so that information from previous generations can also be used to derive the covariance matrix for the current. The recommended value for c_μ is described in equation 4.5, where c_1 is the learning rate for the Rank-One update (explained in the next paragraph), μ_{eff} is the *variance effective selection mass* (equation 4.6), which represents the amount of bias provided by the weights, and n is the problem dimensionality (in our case, the number of connection weights in the ANN).

In addition to using learning rates, CMA-ES also uses Rank-One update to improve covariance matrix estimation for small populations. This method iteratively updates the covariance matrix with a single selected step x in order to bias the distribution towards that step. This is achieved by using the equations 4.7 and 4.8. Performing these equations for a sufficient number of iterations will subsequently lead to a Gaussian distribution with an expected value of x.

$$C^g = (1 - c_1)C^{g-1} + c_1 x x^T \tag{4.7}$$

$$c_1 = \frac{2}{(n + 1.3)^2 + \mu_{eff}} \tag{4.8}$$

Instead of performing the Rank-one update on some selected chromosome within the population, CMA-ES aims to instead bias it towards an evolutionary path p_c, which represents the weighted series of mean shifts of the distribution from the start of evolution to the current generation. This can be calculated using equations 4.9 and 4.10, where c_c is the learning rate of p_c, m is the mean, and σ is the step-size. The final Rank-one update is given in equation 4.11.

$$p_c^g = (1 - c_c)p_c^{g-1} + \sqrt[2]{c_c(2 - c_c)\mu_{eff}}\frac{m^g - m^{g-1}}{\sigma} \tag{4.9}$$

$$c_c = \frac{4 + \mu_{eff}/n}{n + 4 + 2\mu_{eff}/n} \tag{4.10}$$

$$C^g = (1 - c_1)C^{g-1} + c_1 p_c p_c^T \tag{4.11}$$

CMA-ES incorporates both Rank-one and Rank-μ updates of the covariance matrix in order to combine their benefits. Equation 4.12 provides the complete equation for achieving this.

$$C^g = (1 - c_1 - c_\mu)C^{g-1} + c_1 p_c p_c^T + \sum_{i=1}^{\mu} w_i((x_i - m)(x_i - m)^T) \tag{4.12}$$

Updating the Step-Size

The step-size of a distribution defines its scale, and is important as it allows the effective adaptation between exploration and exploitation of a search space. A problem with only

updating the covariance matrix and mean is that they do not explicitly define the scale of the distribution, but instead do so implicitly by expanding the distribution towards the better performing individuals of the current generation, and by phasing out information from previous generations. This results in change rates in the step-size that are too slow to achieve evolution times competitive with other NE algorithms [Han05].

Thus, in order to achieve faster evolution, the scale of the distribution also needs to be evolved in tandem with both the covariance matrix and the mean. This is achieved with equations 4.13 - 4.16, where σ^g is the step-size of the current generation, $E\|N(0,I)\|$ is the expectation of $N(0,I)$, p_σ is the conjugate evolution path, c_σ is the learning rate, and d_σ is a damping factor. Additionally, $C^{-\frac{1}{2}}$ can be calculated with BDB^T, where B is a matrix of Eigen vectors for C, D is a diagonal matrix of square rooted Eigen values for C, and $BDDB^T$ is the Eigen decomposition of C.

$$\sigma^g = \sigma^{g-1} exp(\frac{c_\sigma}{d_\sigma}(\frac{\|p_\sigma^g\|}{E\|N(0,I)\|} - 1)) \tag{4.13}$$

$$p_\sigma^g = (1-c_\sigma)p_\sigma^{g-1} + \sqrt{2}c_\sigma(2-c_\sigma)\mu_{eff}C^{(g-1)^-\frac{1}{2}}\frac{m^g - m^{g-1}}{\sigma^{g-1}} \tag{4.14}$$

$$c_\sigma = \frac{\mu_{eff}+2}{n+\mu_{eff}+5} \tag{4.15}$$

$$d_\sigma = 1 + 2max(0, \sqrt{2}\frac{\mu_{eff}-1}{n+1} - 1) + c_\sigma \tag{4.16}$$

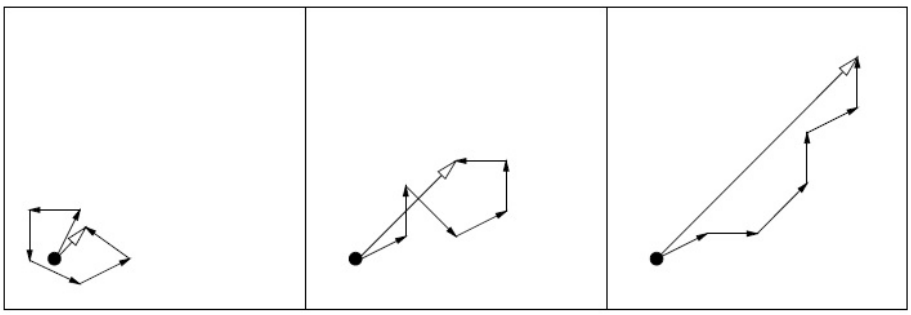

Figure 4.6: *Step-size evolution in CMA-ES. If previous steps are correlated as seen in the right image, then the step-size increases, which allows for the population to reach a specific region in fewer steps. If the previous steps are uncorrelated(left and middle), then the step-size decreases. This allows for finer exploitation of the area, as well as preventing steps that cancel each other out as seen in the left image (Source: [Han05]).*

Using equations 4.13 - 4.16, the step-size increases when recent previous steps are more correlated, and decreases when they are uncorrelated, as shown in figure 4.6. This helps speed up evolution as it decreases the number of steps required for the distribution to shift towards a well performing region, and allows the distribution to perform a more refined search when it is already within a good region.

Algorithm 4: NEAT

Initialize population P without hidden nodes;
Produces species $P_1 ... P_n$ from P with Speciation;
Evaluate(P);
while *Termination criteria not met* **do**
    foreach P_i **do**
        Assign a rank based on average fitnesses;
        Calculate number of offspring α_i to be created for P_i;
        Select parents S_i from P_i using some selection operator;
        Produce offspring O_i from S_i using crossover;
        Mutate O_i using some mutation operator;
    end
    Set P to O;
    Produces species $P_1 ... P_n$ from P with Speciation;
    Evaluate(P);
end
return P_{best}

4.1.4 NEAT

NEAT is a NE algorithm that evolves both the weights and topology of ANNs. It achieves this by using concepts such as a unique chromosome representation, complexification, speciation, historical markings, and unique mutation and crossover operators. The NEAT algorithm is detailed in algorithm 4. The rest of this section will elaborate on the specific operators and workings of NEAT.

Chromosome Representation

Each chromosome in NEAT is represented by two vectors, one for nodes and another for connections. The node vector lists all the nodes of the network. Each node is given an identifier and a type denoting whether it is an input, hidden, or output node.

The connections vector details all the connections between the nodes of the network. The attributes for these connections are as follows:

- **Input node id**: Specifies the node that sends data through this connection.
- **Output node id**: Specifies the node that receives input data from this connection.
- **Weight**: Specifies the weight for the output node to use when calculating its net input signal.
- **Enabled**: A boolean signifying whether or not the connection is currently active in the network. Inactive connections are not used when running the ANN. All connections are initially enabled, and are only disabled when nodes are added.
- **Innovation number**: An integer used to denote the historical origin of the connection.

Figure 4.7 shows an example chromosome for NEAT, and the resultant ANN obtained from the gene information.

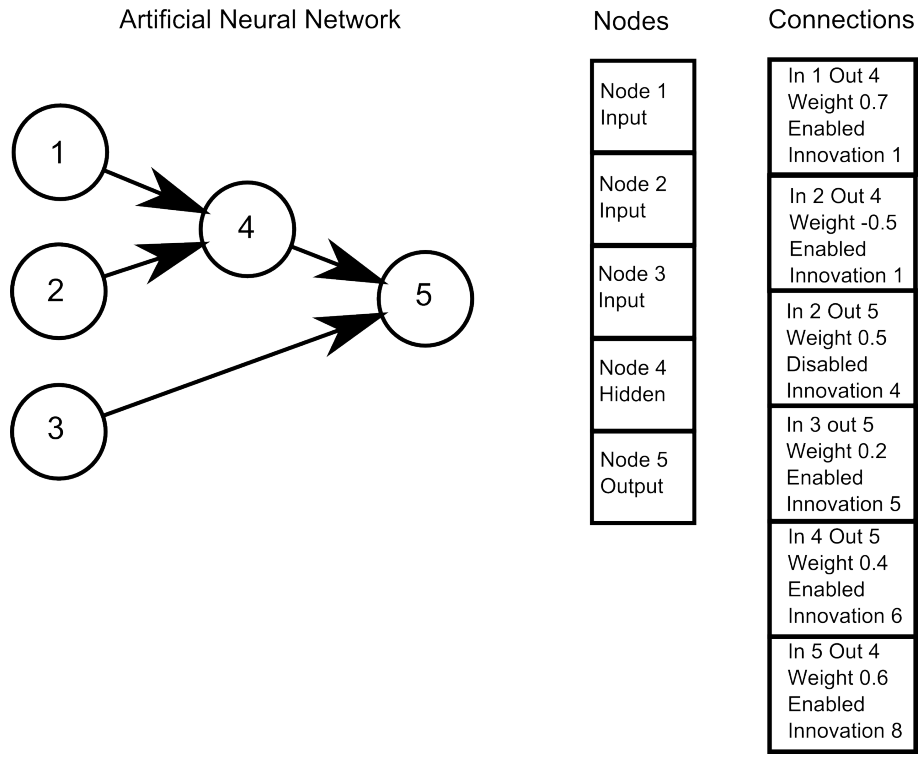

Figure 4.7: *A NEAT chromosome and the ANN that it represents. Note that since the connection with innovation number 4 is disabled, it is not used in the network.*

Historical Markings

Since NEAT allows chromosomes with varying topologies, it is important to have some mechanism to determine the similarity between the chromosomes in order to evolve them. One possible way to do this is through topological analysis. However, this is computationally expensive. NEAT approaches this problem by instead using historical markings.

Historical markings are *innovation numbers* assigned to each connection gene, and denote that gene's historical origin. These markings are assigned according to a *global innovation number*. Every time a new connection is formed using the NEAT mutation operator, the global innovation number is incremented and assigned.

When comparing two chromosomes, the connections that have matching innovation numbers are considered matching connections, whereas the ones that do not match are termed either disjoint or excess, depending on whether the innovation number is smaller than the largest innovation number of the other network (disjoint) or larger (excess). This allows one to determine how different two chromosomes are, as well as provide information on how to perform crossover.

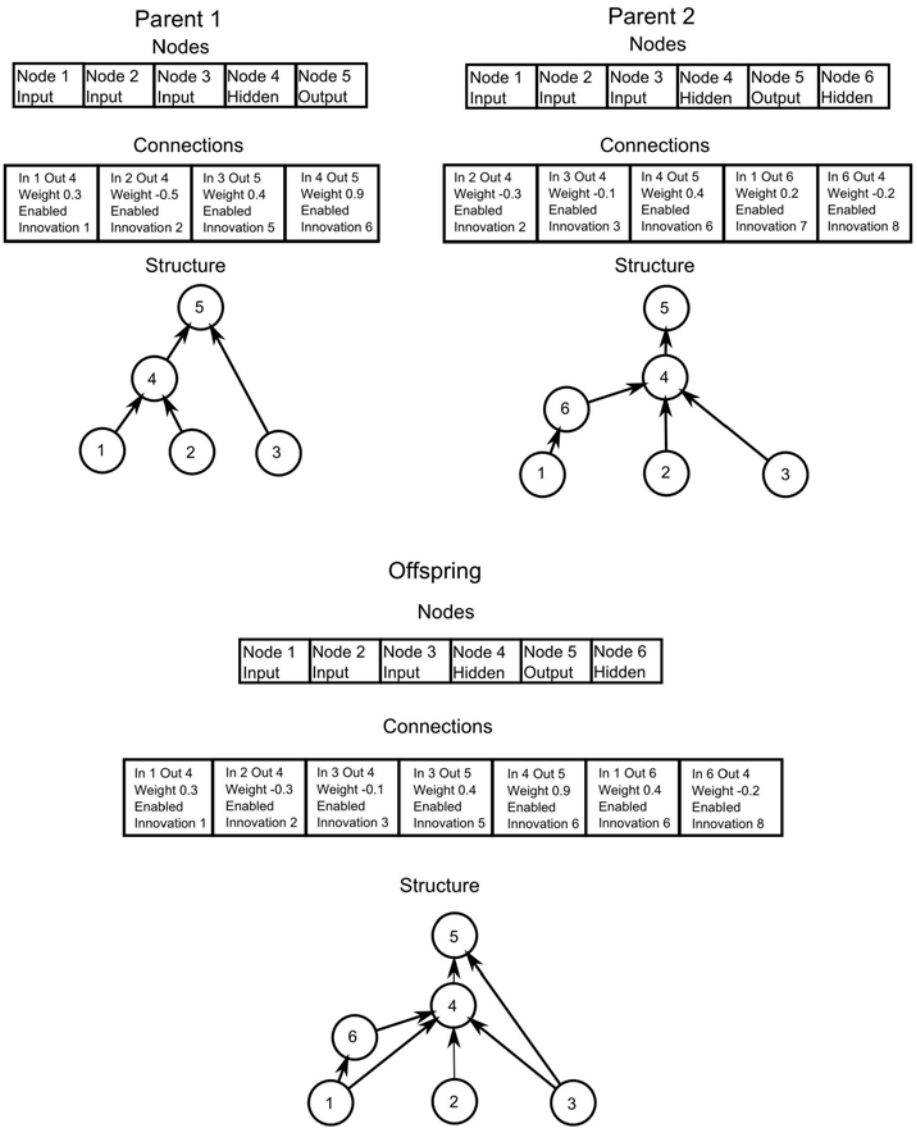

Figure 4.8: *Crossing over neat chromosomes when the fitnesses of the two parents are equal. In the case where the fitnesses are not equal, the topology is inherited from the more fit parent. (Image source: [SM02])*

Crossover

Crossover in NEAT allows for inheritance of both topology and weights. When creating a child from two parents, the topology is inherited from the fitter parent, with the weights of the matching connections inherited randomly. In the case where parents have equal fitnesses, the topology is inherited from both parents. This is done by inheriting the union of the connections and nodes from both parents, with the matching connections inheriting weights randomly, as shown in figure 4.8.

In order to select parents for crossover, a selection operator is used. We have implemented 4 such operators for our system, which are explained in section 4.4. Details and motivations for the specific operator used in our experiments are further discussed in chapter 7. Another aspect of crossover that also needs to be dealt with is *crossover*

probability. This refers to the chance of one of the parents moving on unchanged to the next generation instead of producing offspring.

Mutation

There are three types of mutation used in NEAT. The first is *weight mutation*, where a mutation operator (from section 4.3) is used to perturb the value of the weight.

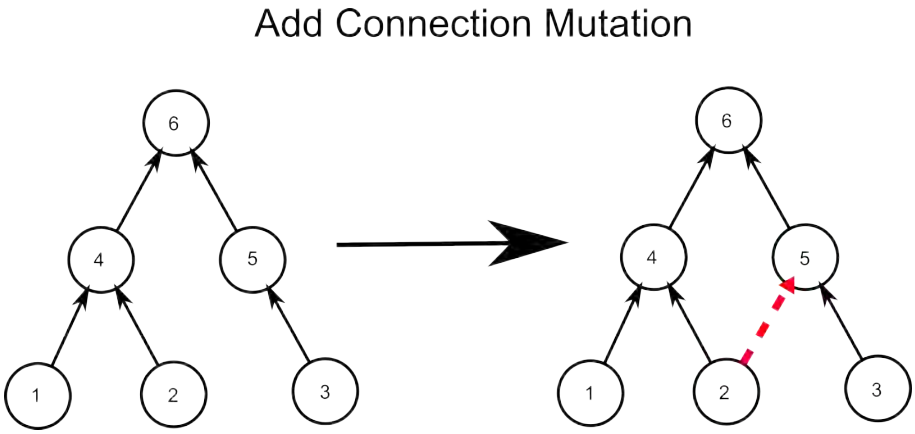

Figure 4.9: *Add connection mutation operation of NEAT. A connection is added between nodes 2 and 5 in this figure. This new connection is assigned a new innovation number, and added to the chromosome's connection genotype.*

The second is the *add connection* mutation, where connections are added to the existing topology, with each unconnected pair of nodes having a connection probability (figure 4.9). As we are only dealing with Feed-Forward Neural Networks, we prevented this mutation from forming loops within our networks, as it would result in Recurrent Neural Networks, which are beyond the scope of the current work.

The last type of mutation is the *add node* mutation, where a new neuron is incorporated into the ANN. In order to add a neuron, a random connection is selected. One new node, representing the neuron, and two new connections are then created, with these connections linking the new node to the in-nodes and out-nodes of the previous connection (figure 4.10). The previous connection is then disabled. The weight of the outgoing connection from the new node is initialized to the weight of the previous (now disabled) connection, whereas the weight of the incoming connection to the new node is initialized to 1. This reduces the initial effect of the mutation [SM02].

Complexification

The operators discussed so far deal with how to change the topology of the chromosomes in NEAT. Another consideration is how to initialize them. Each chromosome could be assigned a random ANN topology. However, this often leads to network topologies with many superfluous nodes and connections, an undesirable trait as minimal network structures allow for significantly faster evolution times [SM02]. Furthermore, smaller networks also allow for less degrees of freedom, thus, reducing the risk of overfitting.

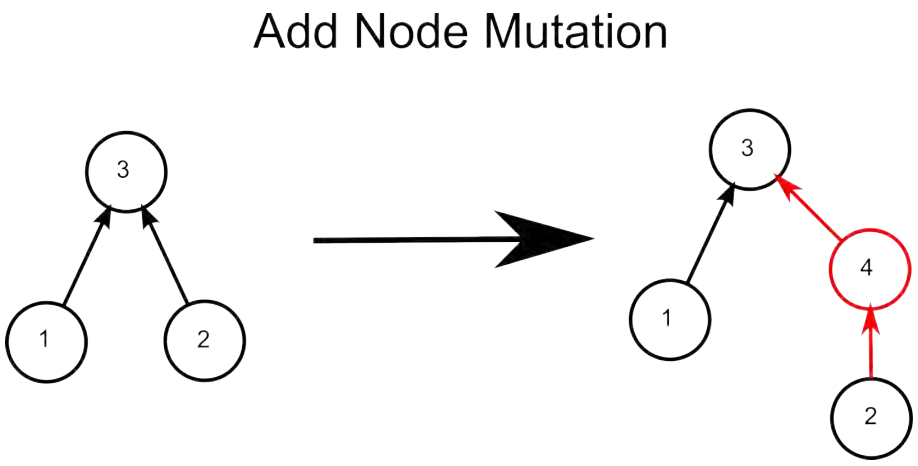

Figure 4.10: *Add node mutation of NEAT. The connection between nodes 2 and 3 is selected and disabled in this figure, with a new node 4 being created. Two new connections connect node 4 to nodes 3 and 2.*

NEAT combats this by initializing the networks without hidden nodes. This minimalist approach allows the chromosomes to remain small, with topological innovations justified because they will only persist if they provide an improvement over previous chromosomes. This approach is termed *Complexification*.

Speciation

A problem with modifying ANN topologies is that the topological innovations often lower task performance in the short term, resulting in them being rejected and not being allowed time to optimize further. In order to protect these innovations, NEAT uses *Speciation*.

Speciation separates the population into species, determined by similarity between the topologies of the chromosomes. Crossover occurs only within these species, therefore allowing structural innovation preservation by providing time for optimization. To create these species, a random chromosome is selected to represent each species of the previous generation. The current population of chromosomes are then compared for similarity with these representatives, and assigned to the first matching species. This prevents them from simultaneously existing in multiple species. If a chromosome is not sufficiently similar to an existing representative, a new species is created with that chromosome as the representative. Chromosomes in the first generation are all assigned to the same species, as Complexification causes them to have the same initial structure.

A metric termed *compatibility distance* is used to determine similarity between chromosomes. This is calculated using $\frac{c_1\epsilon+c_2\delta}{N} + c_3\omega$, where c_1, c_2, and c_3 are constants, N is the number of connections in the larger of the chromosomes being compared, ω is the average of the differences between the weights of matching connections, δ is the number of disjoint connections, and ε is the number of excess connections. In order for a chromosome to be similar to another, the compatibility distance has to be below a pre-defined *compatibility threshold*. As this threshold is problem specific, preliminary tests are required to determine the best performing value. These tests, and our chosen threshold for our experiments, are discussed in section 7.2.1.

In order to determine how many offspring each species should generate, the species are ranked based on their average fitnesses. The number of offspring that each species produces is based on these ranks, with better ranked species producing a higher number of offspring. The number of offspring the i_{th} ranked species (assuming the species are ranked best to worst) generates can be obtained with $\alpha_i \approx \frac{\beta-(i-1)}{\gamma}\lambda$, where β is the number of species, λ is the population size, $i\epsilon[1,\beta]$ is the rank of the species, and $\gamma = \frac{\beta(\beta+1)}{2}$ is used to normalize the number of offspring generated against the number of species.

4.2 GA Crossover Operators

Both CNE and and ESP use existing GA crossover operators. However, there is no indication as to how well each operator will perform in the specific context of controlling crowd simulations. For this reason, we implement and test ten well established crossover operators to investigate which operator performs best.

The notation for formulae in this section refer to y_{ij} as the j_{th} dimension of the i_{th} offspring, x_{ij} as the j_{th} dimension of the i_{th} parent, $U(a,b)$ as a uniform distribution with the domain $[a,b]$, and $N(a,b)$ as a normal distribution with a mean of a and a standard deviation of b.

4.2.1 One-point Crossover

One method of combining chromosomes is to use discrete crossover operators. One-point crossover [Hol75] is one such operator, where offspring are created from 2 parents by inheriting all the genes before a specific position r from one parent, and all the genes after that position from the other. The position is randomly generated by sampling from a uniform integer distribution between 1 and the number of genomes in one of the parents. Equation 4.17 is used for generating the offspring, where $r \sim U[1,n]$, with n being the dimensionality of the search space.

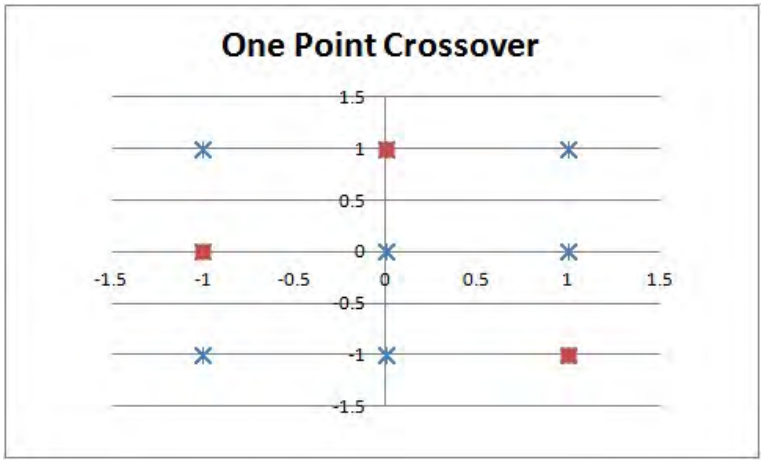

Figure 4.11: *One point crossover performed in two dimensions in a population of three parents. Offspring are denoted in blue whereas parents are denoted in red.*

$$y_{ij} = \begin{cases} x_{1j} & \text{if } j <= r \\ x_{2j} & \text{if } j > r \end{cases} \tag{4.17}$$

4.2.2 Two-point Crossover

Two-point crossover [ECS89, DJ75] is another discrete crossover operator that uses 2 parents. Like one-point crossover, offspring also inherit genes from either one parent or the other depending on the position of the gene. However, it differs in that instead of one position being generated, two are instead (r_1 and r_2). The offspring will thus inherit the genes between the two generated positions from one parent, and the genes outside the two positions from the other. Mathematically, offspring are generated using equation 4.18, where $r_1 \sim U(1, n)$ and $r_2 \sim U(1, n)$, with n being the dimensionality of the search space and $r_2 > r_1$. The offspring generated by two-point crossover is identical to that of one-point crossover in two dimensions (figure 4.11), reason being that there is only one position (not including the start and end) within a two-dimensional chromosome where points can be generated. In n dimensions, the offspring that one-point crossover generates are a subset of two-point crossover, with two-point crossover generating more diverse offspring when $n > 2$.

$$y_{ij} = \begin{cases} x_{1j} & \text{if } j <= r_1 \\ x_{2j} & \text{if } j > r_1 \text{ and } j <= r_2 \\ x_{1j} & \text{if } j > r_2 \end{cases} \tag{4.18}$$

4.2.3 Uniform Crossover

Uniform crossover [Sys89] is another discrete crossover operator, where offspring are once again generated by inheriting genes from one of two parents depending on the position of the gene (figure 4.12). In Uniform crossover, the parent from which the gene at specific position j is determined by sampling r_j from a uniform distribution between 0 and 1. If r_j is smaller than or equal to 0.5, the gene is inherited from the first parent, whereas if it is larger than 0.5, the gene is inherited from the other.

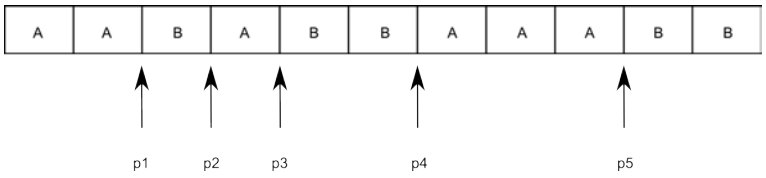

Figure 4.12: *In uniform crossover, points specify when the offspring's gene sequence shifts from one parent's to another's.*

$$y_{ij} = \begin{cases} x_{1j} & \text{if } r_j <= 0.5 \\ x_{2j} & \text{if } r_j > 0.5 \end{cases} \tag{4.19}$$

Offspring are generated using equation 4.19, where $r_j \sim U(0, 1)$. The offspring generated by uniform crossover in two dimensions are identical to one- and two-point crossover

(figure 4.11). Thus, the rules that apply to the relation between one- and two-point crossover also apply to uniform crossover. Uniform crossover is able to generate more diverse offspring than one-point crossover when $n > 2$, and than two-point crossover when $n > 3$.

4.2.4 Simulated Binary Crossover

A problem with using discrete operators for continuous problems is that the offspring may not cover the search space adequately. Simulated binary crossover [ADA94] aims to alleviate this by approximating one-point crossover (in the context of evolving binary strings) in continuous domains. This is achieved by designing the operator to produce offspring in a similar probability distribution to that of one-point crossover [ADA94].

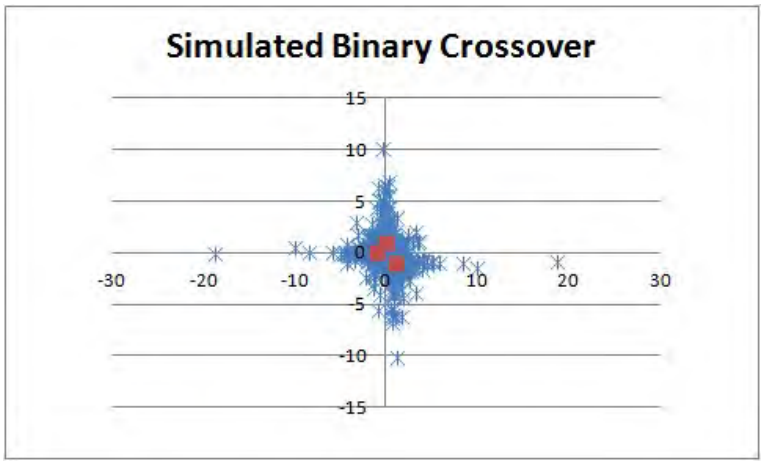

Figure 4.13: *Simulated binary crossover approximates both the distribution and the search power of one-point crossover when used with binary strings.*

$$
\begin{aligned}
y_{1j} &= 0.5((1 + w_j)x_{1j} + (1 - w_j)x_{2j}) \\
y_{2j} &= 0.5((1 - w_j)x_{1j} + (1 + w_j)x_{2j}) \\
w_j &= \begin{cases} (2r_j)^{\frac{1}{\eta+1}} & \text{if } r_j <= 0.5 \\ (1/2(1 - r_j))^{\frac{1}{\eta+1}} & \text{if } r_j > 0.5 \end{cases}
\end{aligned}
\tag{4.20}
$$

The shape of the distribution is shown in figure 4.13. One should note that this distribution is symmetrical around the parents, and thus avoids biasing the search process. To generate offspring, one uses equation 4.20, where $r_j \sim U(0, 1)$ and $\eta > 0$. η is the distribution index that determines how close offspring are generated relative to their parents. Deb and Agrawal [ADA94] recommends $\eta = 1$ as a setting that generally works well.

4.2.5 Arithmetic Crossover

$$
y_{ij} = \sum_{k=1}^{n} w_k x_{kj} \tag{4.21}
$$

Another simple continuous crossover operator is arithmetic crossover [Mic96] (figure 4.14), which produces an offspring that is the weighted average of n selected parents. Offspring are generated from n parents using equation 4.21, where w_k is a random number between 0 and 1, and $\sum_{k=1}^{n} w_k = 1$.

A problem with multi-parent operators is that finding the ideal number of parents would result in too many different test scenarios, leading to us exceeding our time constraints. For this reason, we merely use the minimum number of allowed parents, which for this operator is 2.

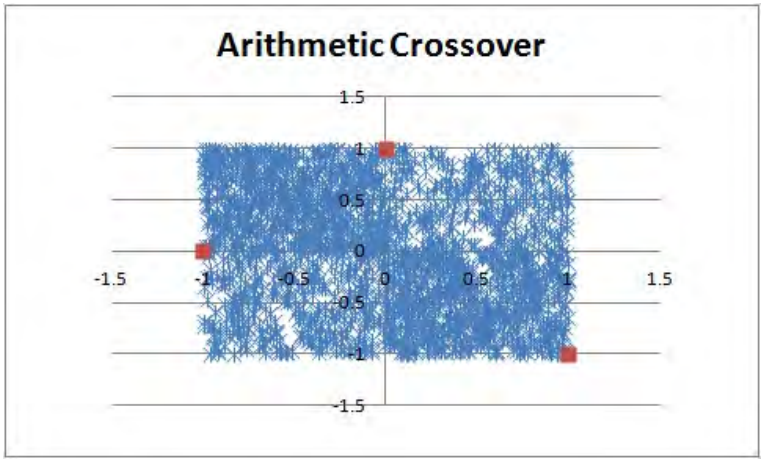

Figure 4.14: *Arithmetic Crossover. Offspring are generated uniformly in a hyper cube defined by the min and max values of the parents in each dimension.*

4.2.6 Blend Crossover(BLX-α)

A problem with arithmetic crossover is that offspring are only generated between the parents. Thus, mutation is needed to search in areas not bounded by the parents. Blend crossover (BLX-α) [ES93] remedies this by expanding the hypercube so that offspring can also be generated in an area around that defined by the parents. This is achieved using equation 4.22, where $w_j = (1 + 2\alpha)U(0, 1) - \alpha$, with the recommended value for α being 0.5 [ES93].

$$y_{ij} = (1 - w_j)x_{1j} + w_j x_{2j} \tag{4.22}$$

BLX-α has the property that the larger the distance between the parents, the larger the distance between the offspring and the parents. The offspring are randomly generated within the range $[x_{1j} - \alpha(x_{2j} - x_{1j}), x_{2j} + \alpha(x_{2j} - x_{1j})]$ (figure 4.15).

4.2.7 Heuristic Crossover

Heuristic crossover [Wri91] is an operator that takes the fitnesses of the parents into account, by creating offspring in the direction of the better performing parent (figure 4.16). It generates one offspring from two parents using equation 4.23, where the fitness of x_{1j} is better or equal to the fitness of x_{2j}.

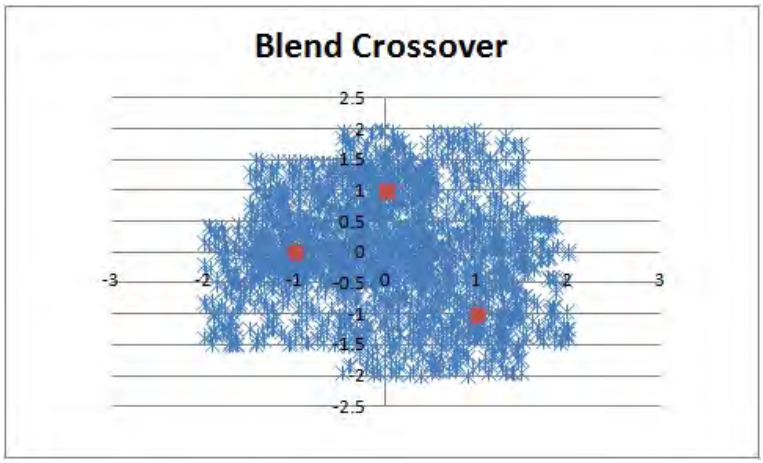

Figure 4.15: *Blend Crossover(BLX-α). Given two parents, offspring are generated uniformly in a hypercube around the two selected parents. This hypercube has the dimensions* $[x_{1j} - \alpha(x_{2j} - x_{1j}), x_{2j} + \alpha(x_{2j} - x_{1j})]$. *Thus, with the recommended* $\alpha = 0.5$, *each dimension is twice as long as the distance between the two parents in that dimension.*

$$y_{ij} = U(0,1)(x_{1j} - x_{2j}) + x_{1j} \tag{4.23}$$

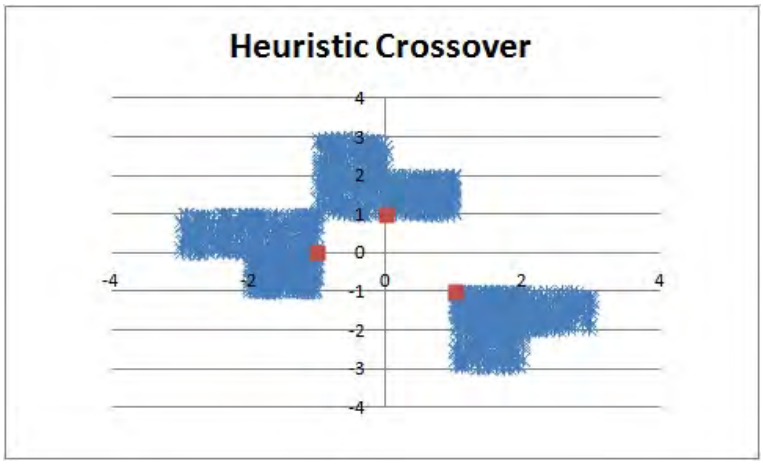

Figure 4.16: *Heuristic Crossover assuming all three parents have the same fitness. Heuristic crossover searches in the area of the better performing parent, thus applying a significant amount of selective pressure.*

A drawback of this operator is that it will greatly bias the population distribution towards the direction of the better performing parent, impeding exploration.

4.2.8 Uni-modal Normal Distribution Crossover

Most of the previously discussed crossover operators generate offspring in a uniform manner. Uni-modal normal distribution crossover [OK97], on the other hand, generates offspring in a multi-variate Gaussian distribution (figure 4.17) defined by three parents.

Offspring can be produced using equation 4.24, where x^p is the midpoint between parents x_1 and x_2 and defines the centre of the distribution; d is the difference vector $x_1 - x_2$ and is used as one of the principal axes of the ellipsoid defining the distribution; n is the dimensionality of the search space; and D is the distance from parent x_3 to its projection on d which is used to determine the scale of the distribution.

$$y_i = x^p + \xi d + D \sum_{k=1}^{n-1} \eta_k e_k \tag{4.24}$$

The orthogonal basis vectors e_i span the subspace orthogonal to the vector space defined by d, and represent the other principal axes of the ellipsoid. The variables $\xi \sim N(0, \sigma_\xi^2)$ and $\eta_i \sim N(0, \sigma_\eta^2)$, where $N(0, \sigma^2)$ is a random number sampled from a Gaussian distribution with a mean of 0 and a variance of σ^2, define the standard deviation along d and the other principal axes respectively. Ono and Kobayashi [OK97] suggest that $\sigma_\eta = \frac{0.35}{n}$ and $\sigma_\xi = \frac{1}{2}$.

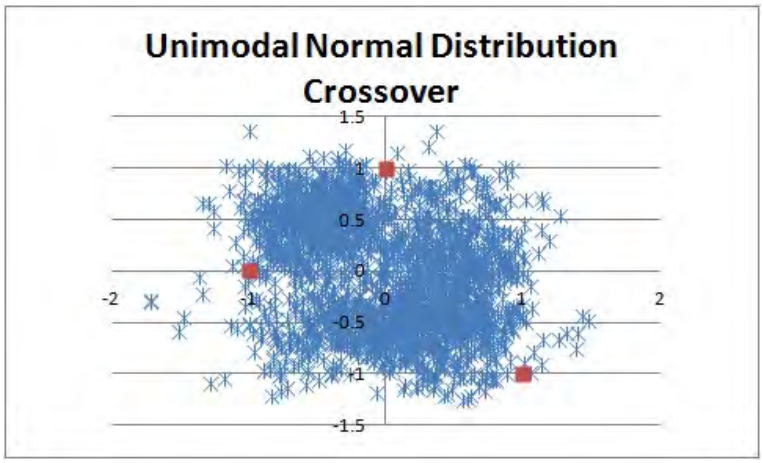

Figure 4.17: *Uni-modal normal distribution crossover. Offspring are generated in normal distributions defined by the three parents.*

4.2.9 Parent Centric Crossover

Parent centric crossover [DPAM02] is an extension to uni-modal normal distribution crossover that generates offspring in a normal distribution centred around one of the three selected parents (named the female). The effect of this is that the evolutionary process will search near the parents, as opposed to near the centre of mass of two of the parents as in the uni-modal normal distribution crossover. Additionally, as the parent from which to generate the offspring around is selected randomly, the operator essentially ends up searching around all the parents (figure 4.18).

$$y = x_p + w_\zeta |d^{(p)}| + D \sum_{i=1, i \neq p}^{\mu} w_\mu \eta_k e^i \tag{4.25}$$

From n parents, offspring can be generated using equation 4.25, where x_p is the female parent chosen randomly for each offspring and is used as the centre of the distribution;

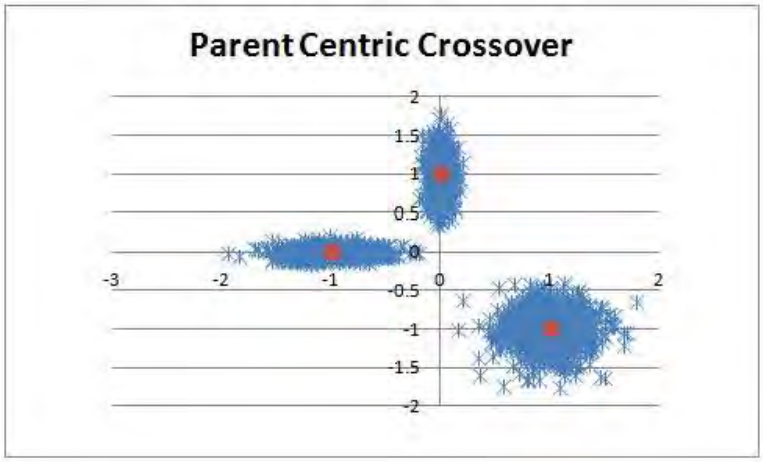

Figure 4.18: *Parent centric crossover generates offspring in a normal distribution around a randomly selected female parent.*

$d^{(p)}$ the direction vector $x_p - g$ used as one of the principal axes of the ellipsoid, with g being the mean vector of the μ parents; D being the average of distances of all the non-female parents from $d^{(p)}$, and is used to define the scale of the distribution; e^i being the $\mu - 1$ orthonormal basis that span the subspace orthogonal to $d^{(p)}$ used to define the other principal axes of the distribution; and w_ζ and w_μ are zero-mean Gaussian distributed variables with variance of σ_η^2 and σ_μ^2, respectively. As 3 is the minimum number of parents needed for this operator, we use that number in our experiments. In order to modify the distribution of the offspring, one can modify σ_η^2 and σ_μ^2, which will change the variance of offspring along d^p and all other orthogonal directions respectively.

4.2.10 Laplace Crossover

Another parent-centric crossover operator is Laplace crossover [DT07], where offspring are generated around the parents according to a Laplace distribution (figure 4.19), a distribution where the probability density increases exponentially towards the mode (figure 4.20). This differs from parent centric crossover which generates offspring in a Gaussian distribution around the parents. Specifically, the Laplace distribution has fatter tails than the Gaussian, resulting in the exploration of a larger section of the search space.

$$\begin{aligned} y_1 &= x_1 + \beta|x_1 - x_2| \\ y_2 &= x_2 + \beta|x_1 - x_2| \\ \beta &= \begin{cases} a + b\ log(2u) & \text{if } u <= 0.5 \\ a - b\ log(2 - 2u) & \text{if } u > 0.5 \end{cases} \end{aligned} \tag{4.26}$$

In Laplace crossover, two offspring are generated for a pair of parents using equation 4.26, where $u \sim U(0, 1)$; a is a constant which determines the location of the distribution; and b is a constant that determines how close to the parents the offspring are generated.

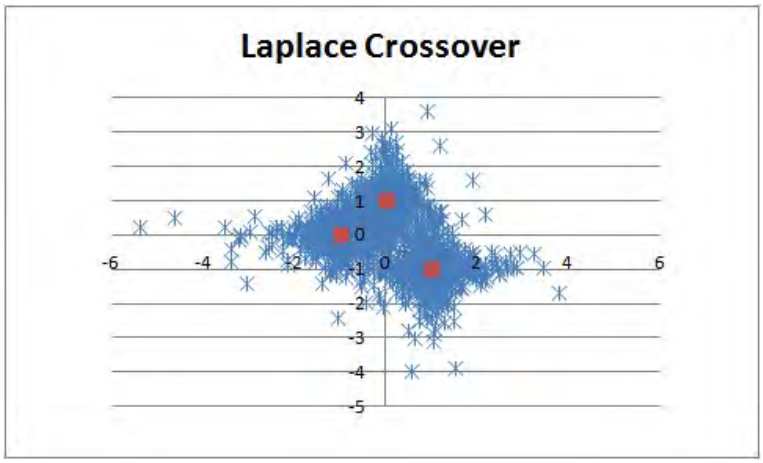

Figure 4.19: *Laplace crossover generates offspring around parents in a Laplace distribution.*

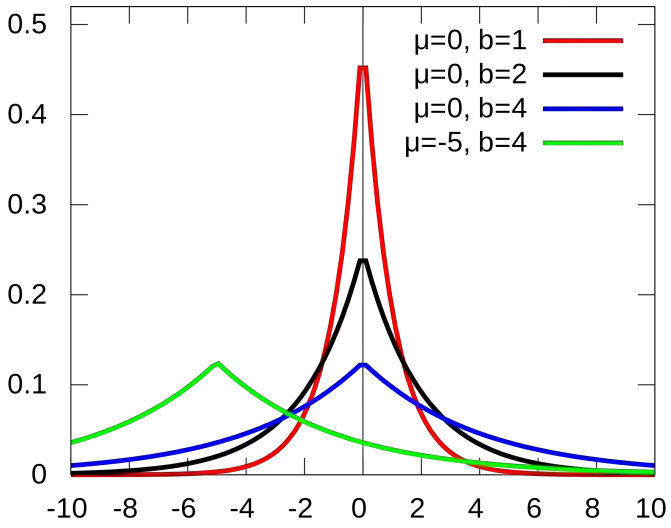

Figure 4.20: *The Laplace distribution with differing values of the mode μ and b (Source: [Lap]).*

4.3 GA Mutation Operators

In addition to crossover, mutation is also an important operator for GAs. In environments where sequences of bit-strings are evolved, it is sufficient to merely flip random bits. However, GAs that evolve chromosomes with a continuous representation must approach this differently, typically by adding noise sampled from some distribution to the chromosomes. Much like crossover operators, it is an open question as to which mutation distributions performs better in our use case. Thus, we have implemented three commonly used operators for our system in order to evaluate their performance.

4.3.1 Uniform mutation

One common distribution used for mutation is the uniform distribution. In order to mutate an offspring, values sampled from $U(-\alpha, \alpha)$ are added to the chromosome, with α being problem dependent. One difficulty with using a uniform distribution is that the noise is limited to $[-\alpha;\ \alpha]$, thus it is not always possible to escape from local optima. Another problem is that a large number of mutations are required in order for the operator to not unintentionally bias the search process, as the mean of the mutations needs to be close to 0 in order for the operator to be unbiased.

4.3.2 Gaussian mutation

Another distribution suited to mutation is the normal distribution (figure 4.21), where offspring are mutated using values sampled from $N(0, \sigma)$ (again σ is problem dependent). Using normal instead of uniform distributions has the benefit that fewer mutations are needed for the operator to be unbiased, as the majority of the mutation values are close to zero. Additionally, unlike uniform mutation, the noise values generated are not bound to be within a specific range, thus allowing a better mechanism for escaping local optima.

4.3.3 Cauchy-Lorentz mutation

Finally, we implemented the Cauchy-Lorentz distribution. The shape of this distribution is similar to that of a Gaussian distribution, but with thicker tails and a thinner centre (figure 4.21), resulting in more values generated further away from the peak. This improves the chances of escaping local optima. Values can be generated from this distribution with $C(l, s)$, where the l is the location and s is the scale. An interesting side-note about this distribution is that its mean and standard deviation are undefined. This means that regardless of the number of samples taken, the mean does not converge.

4.4 Selection Operators

Selection is used to determine which chromosomes are permitted to reproduce. This allows the application of selective pressure to the evolutionary process. Much like crossover and mutation operators, there have been no results indicating the relative performance of selection operators when used to control crowds. We have thus implemented four different, well established selection operators in order to investigate their viability. However, we chose not to implement certain operators, such as proportionate selection [Hol75], due to their high selective pressure - an undesirable property for GAs, where lower selective pressure operators are preferred [Bac94].

4.4.1 Linear Rank-based selection

Rank-based selection [Bak85] assigns selection probabilities ($P(x_i)$) to chromosomes based on their fitness ranking within the population.

A linearly increasing approach to this is equation 4.27, where λ is the population size, x_i is the chromosome with the i^{th} best fitness, and $\mu \in [1; 2]$ is the number of

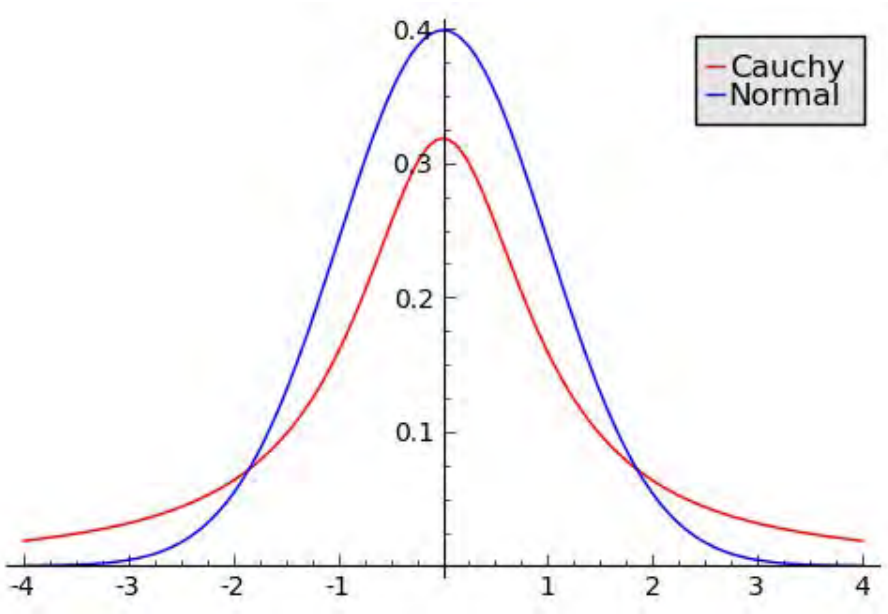

Figure 4.21: *The respective probability densities of Gaussian and Cauchy distributions (Source: [Cau]).*

offspring to be generated for the most fit chromosome, and can be increased or decreased to modify the amount of selective pressure exerted by the operator. We used $\mu = 2$ in our implementation as we found that having a higher selective pressure for this operator allowed for faster evolution.

$$P(x_i) = 2 - \mu + 2(\mu - 1)\frac{\lambda - rank_i}{\lambda} \tag{4.27}$$

4.4.2 Non-Linear Rank-based selection

In addition to using probabilities that linearly increase according to rank, one can also increase the probabilities non-linearly. This has the effect of exploiting more around the best performing individuals as the probability ratio of the better performing individuals significantly outweighs that of weaker individuals. This results in a much higher amount of selective pressure compared to the linear counterpart.

Equation 4.28 is used to calculate the selection probabilities of chromosomes in our non-linear rank-based selection implementation, where $v\epsilon[0, 1]$ denotes the probability of selecting the fittest parent. Higher values of v thus drive more exploitation around the highly ranked chromosomes.

$$p(x_i) = v(1 - v)^{rank_i} \tag{4.28}$$

4.4.3 Tournament selection

Tournament selection [Bri81] runs a single-elimination tournament between n selected chromosomes of the population, with the fitnesses being used to determine the winner between pairs of chromosomes. The best performing α chromosomes are then selected for

reproduction. Tournament selection has the advantage that it reduces selective pressure because the n chromosomes are randomly selected. However, this is only the case if n is small. In the extreme, where n is equal to the population size, the best individuals will always be selected, resulting in very high selective pressure.

4.4.4 Boltzmann selection

The distributions of the above selection operators remain static throughout. Boltzmann selection [MT93] adopts a different approach where the operator gradually increases selective pressure throughout the evolutionary process. This is achieved by using the same thermo-dynamics principles as Simulated annealing [AK88], with the parent selection probabilities calculated using equation 4.29, where $t \in [0, 1]$ is the temperature parameter, and is decreased over the generations. We use the rank of the chromosomes as opposed to their fitness, as it has been found that using fitnesses results in poor performance in GAs [MT93], as it essentially becomes a scaling method for proportionate selection [Bac94].

$$p(x_i) = \frac{1}{1 + e^{rank_i/t}} \tag{4.29}$$

The distribution of Boltzmann selection is such that the selection probability is similar for most chromosomes when t is close to 1, whereas when t is close to 0, better ranked chromosomes have a significantly higher probability of being selected.

4.4.5 Elitism

Elitism [DLJD00] can also be classified as a selection technique. In contrast to the previous methods, it does not deal with selecting chromosomes for reproduction, but instead ensures that the best n chromosomes from the current generation survive to the next. Elitism is thus typically used to apply additional selective pressure, with the benefit of possibly resulting in more efficient evolution. However, it may also result in premature convergence.

4.5 Multiple ANNs per candidate solution

It is often the case that there are multiple species of agents within a single crowd simulation. One example is *The Battle of Helms Deep* scene in the film *The Lord of the Rings: The Two Towers*, where orcs, humans, elves, and dwarves are present, each representing a different species. As different species often behave and perceive differently, it is not sufficient to use a single ANN to control all agents. We remedy this by allowing multiple ANNs to be evolved per simulation. The details of how we separate these ANNs is discussed in section 5.1.4.

The evolution of multiple ANNs requires a few modifications to NE. One required change is extending the chromosome representation. This is trivial for both CNE and CMA-ES as one merely needs to extend the vector to account for multiple ANNs (figure 4.22). ESP can be changed by adding more sub-populations, with each sub-population now addressing a specific ANN in addition to a neuron position, as seen in figure 4.23.

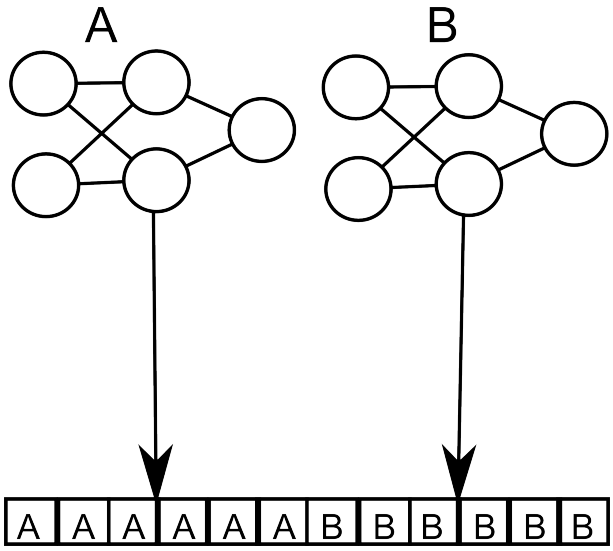

Figure 4.22: *The weights of ANNs A and B are concatenated into one vector in order to evolve A and B together.*

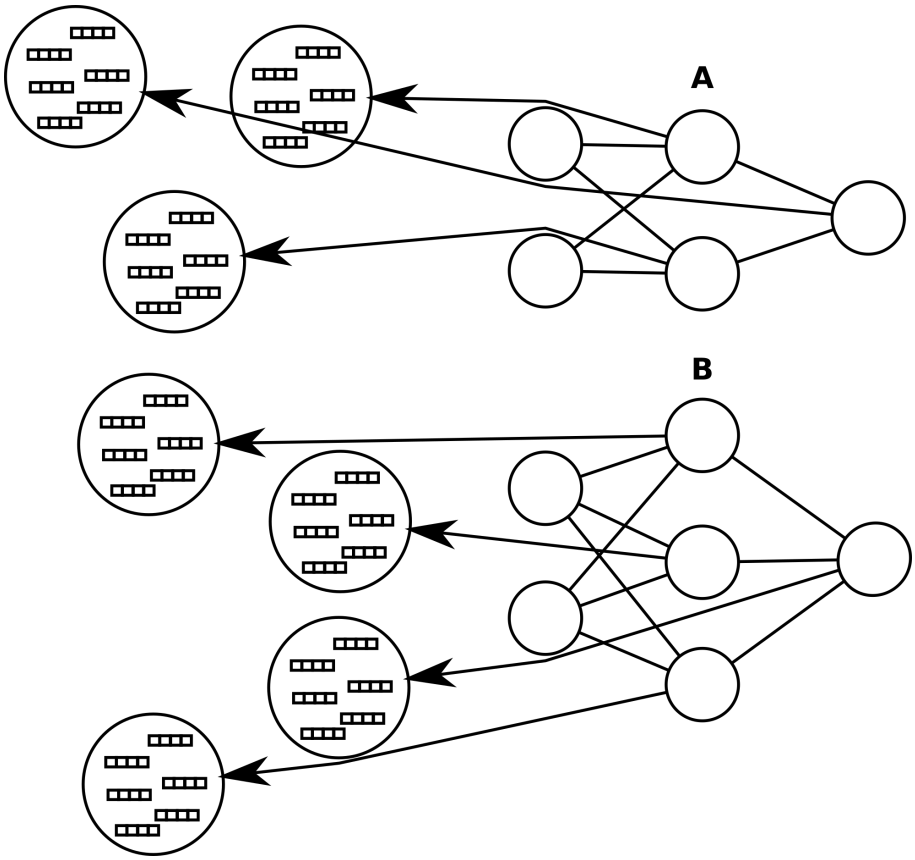

Figure 4.23: *ESP allows for evolving multiple ANNs by providing each sub-population with information of which ANN it belongs to, in addition to its position in the ANN.*

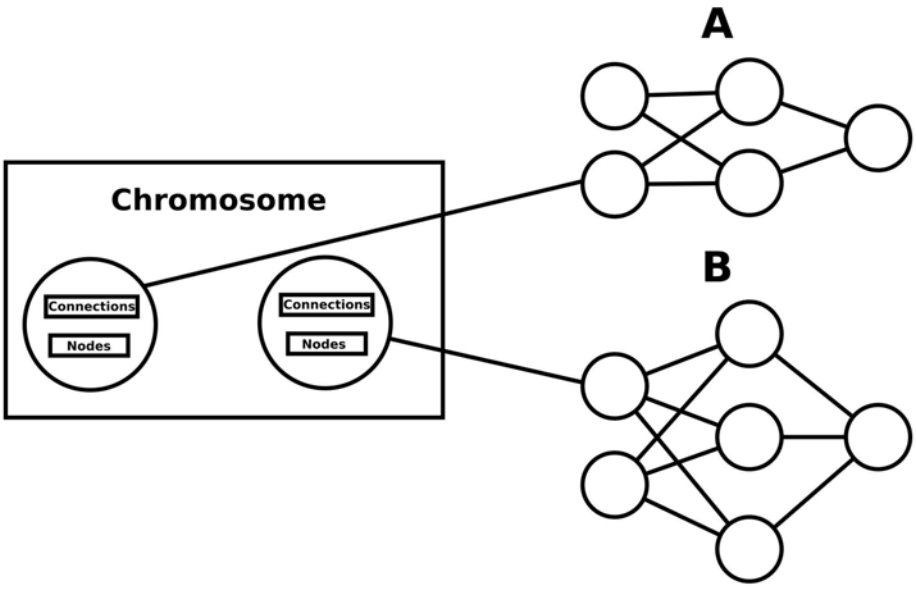

Figure 4.24: *NEAT allows for multiple ANNs by adding a connection and node vector for every additional ANN.*

Such an approach is termed Multi-agent ESP [YM09]. Lastly, with NEAT, we modify the chromosome representation to incorporate an additional vector of connections and nodes for every additional ANN (figure 4.24), which enables us to use a single chromosome to represent multiple ANNs. We ensure that each ANN can only perform crossover with its corresponding ANN in other chromosomes, thus, ensuring that crossover is semantically correct.

In addition to the chromosome representation, we also make a slight modification to NEAT to account for multiple ANNs, by changing the *add node* mutation. Normally, there is a chance of adding one node to the chromosome per mutation. We have modified this so that every ANN in the chromosome has a chance of adding a new node. This is intended to maintain the growth rates for the ANNs so that they remain comparable to using NEAT with a single ANN per chromosome.

One last consideration with NEAT is the calculation of the compatibility distance between chromosomes with multiple ANNs. We achieve this by concatenating the connection vectors of each ANN, and then calculating the compatibility distance normally on these concatenated vectors. This is possible since the innovation numbers are incremented globally, and thus are still unique within the vector.

4.6 Controlling agent behaviour with fitness

The primary aim of this thesis is to evolve ANN controllers for agents within crowd simulations so that the user is able to control the emergent aggregate crowd behaviour. This is achieved by designing the fitness function so that it models the control intended by the user. An example of such a control requirement is having all agents move from one point to another. In order to evaluate a chromosome, the crowd simulation is run for a specific time with the chromosome encoding the agent controllers, and then evaluated

according to the fitness function. If the crowd performs close to the desired behaviour, the ANN controllers receive a good fitness, whereas if the crowd behaves very differently from what is desired, it is assigned a poor fitness. The similarity measures for the user desired control objectives are problem dependent, and the ones we use are fully explained in section 6.2.

We model the desired control as a set of objectives, with each separate control requirement being a separate objective. Examples of objectives include distance to a point, and number of total collisions. Each of these objectives provides closeness rating, and is assigned a weight. The weighted sum of closeness ratings are then used as the final fitness value. The weights for each objective are simulation and objective dependent, with the specific weightings for our simulations detailed in section 6.2. Generally, objectives receive a higher weighting if they are more important, or if the values generated by the objective are small in comparison to the other objectives used in the simulation.

4.7 Summary

This chapter provides implementation details for the various NE algorithms, and their respective operators and parameters, implemented in the NE sub-system. Additionally, it also provides insight to the characteristics, and possible advantages and disadvantages of these.

Information of how we intend to use NE to control the aggregate crowd behaviour, as well as extensions to genotype representations to allow for multiple species within crowds is also provided.

Information and comparisons of the task performance and behaviours of the various discussed aspects in this chapter will be fully discussed and analysed in chapters 7 and 8.

Chapter 5

Simulation and Rendering

The Simulation and Rendering sub-systems are responsible for executing a simulation scenario with a given candidate solution and for rendering it, respectively. We have split these into two separate sub-systems for two reasons: simulations executed during the evolution phase should ideally be as fast as possible in order for the evolution to complete within a reasonable period of time, and rendering is not necessary for the evolution phase, and unnecessarily consumes computational resources.

5.1 Simulation Subsystem

This section aims to elaborate on the design specifics of the simulation sub-system. It does not, however, describe the specific simulations and agents used for our experiments, which instead appear in chapter 6.

5.1.1 Running a simulation

Each simulation scenario consists of a set number of cycles. During each cycle, the simulation state is updated by both the physics engine and any additional rules. Some examples to these rules are checking skeletal simulation state to allow for behavioural transitions, and clamping velocities so that agents do not move faster than their maximum allowed speed. We model the agents' decision-making frequency using *skipThink* [Rey06], where agents make decisions every n cycles, as opposed to every cycle. The reasoning behind this is twofold: it is more efficient, as the ANNs and other decision-making processes are run less often; and it helps smooth the behaviours of agents, as they can be somewhat discontinuous if the decisions are performed every simulation cycle.

Each simulation scenario also defines the number of cycles to be run per second when executed in real-time. This is important for both the rendering sub-system and the physics engine. The rendering sub-system requires this information as it uses it to determine how frequently the simulation should be iterated. The physics engine, on the other hand, requires this information as it uses it to determine the time to forward the physical state of the simulation during each simulation cycle.

5.1.2 Bullet Physics

We use Bullet Physics[1] 2.82 as the physics engine in our simulations to perform a variety of tasks, such as collision detection, ray-casting, calculating acceleration and torque forces, and updating agent positions. We chose Bullet as it has excellent documentation and is open source. In order to synchronize Bullet's simulation cycles with the cycles in the simulation sub-system, we advance the bullet world by a fixed duration, calculated using the simulation cycles per second data defined in the simulation scenario.

5.1.3 Controlling Agents with ANNs

In order to control an agent with an ANN, a series of values representing the agent's local perception of the world are computed for the current time step. These values are then passed to the ANN as inputs, which proceeds to evaluate and produce a series of outputs that are returned to the agent. The agent uses these outputs to determine its behaviour for the current time step (figure 5.1).

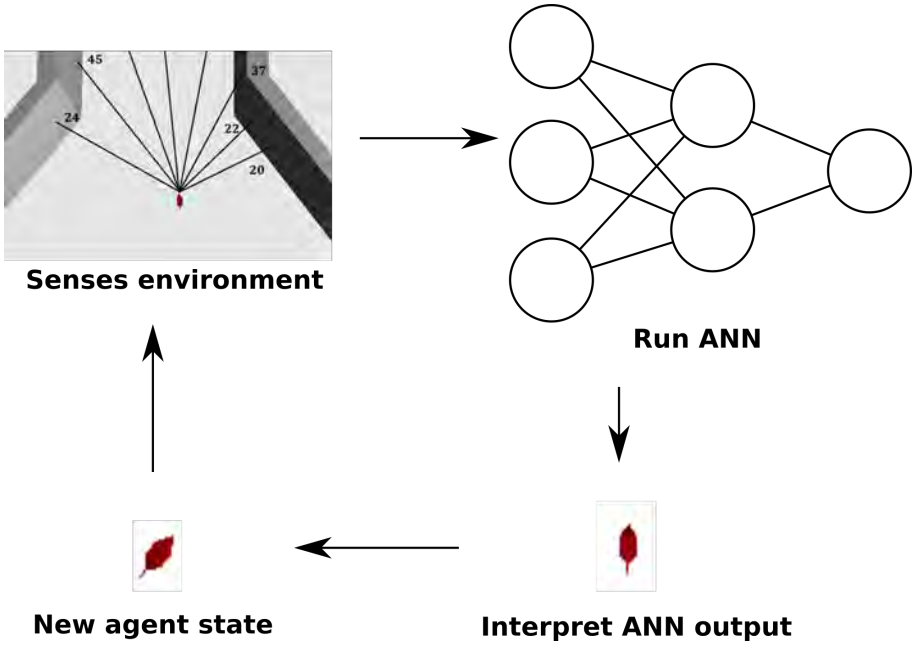

Figure 5.1: *For an ANN to control an agent, the agent provides its perception of the environment as input to the ANN; the ANN then calculates the outputs and returns them to the agent, which then makes a decision on how to act.*

5.1.4 Team architecture

As discussed in section 3.3.5, controller assignments for agents can range from fully homogeneous to fully heterogeneous, with one of the aims of this thesis being to determine what type of setup is most ideal in the context of controlling crowds.

[1] http://bulletphysics.org/

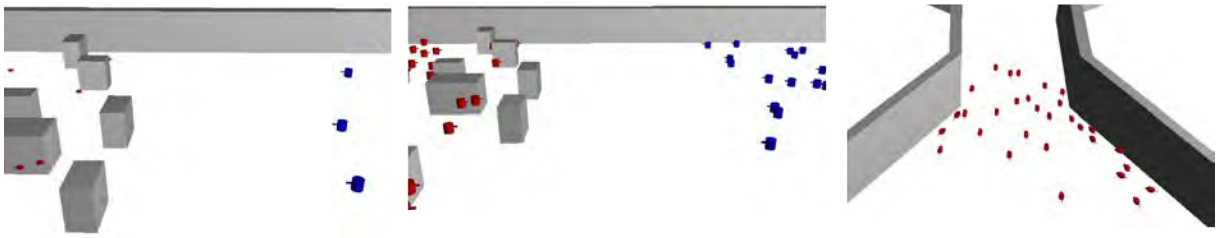

Figure 5.2: *From left to right: two different agent types; two different expected agent behaviours (attacking versus defending); a single group of agents.*

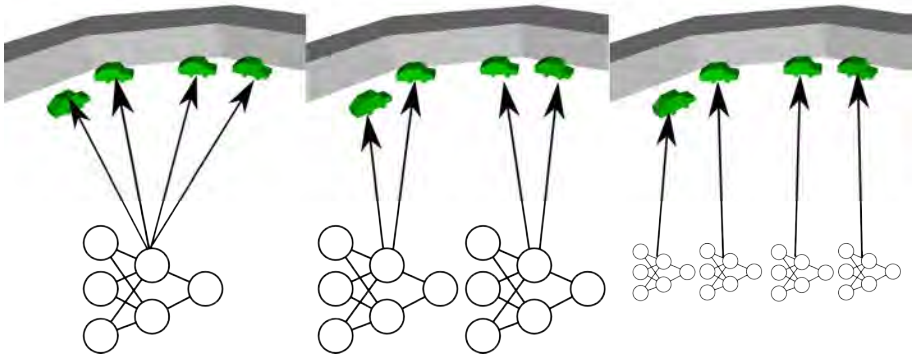

Figure 5.3: *From left to right: Fully homogeneous team setup; A team setup midway between homogeneous and heterogeneous; Fully heterogeneous team setup.*

We achieve this by separating the agents in the simulation into a set of distinct species (figure 5.2). Different species specify either different agent types, such as mice or robots, or different expected agent behaviours, such as defenders or attackers. Each species can then range from having fully homogeneous to fully heterogeneous controller assignment (figure 5.3).

The reason we separated the agents into species is twofold. Firstly, different types of agents, such as mice and robots, require different types and numbers of inputs and outputs, resulting in incompatible ANN topologies. Secondly, different types of behaviours are often required from agents of the same type. For example, in a siege scenario, it is more realistic to have the defending agents adopt passive strategies and the attacking agents aggressive.

5.1.5 Reducing the learning

A problem found in more complex simulations is that agents often struggle in learning how to avoid collisions. To combat this, we implemented a rudimentary collision avoidance algorithm, which reduced the extent of behaviour learning required of the agents. This algorithm only considers environmental obstacles, thus, the agents still need to learn how to avoid each other.

To avoid collisions, rays are cast forward from the front corners of an agent's bounding box (two for two-dimensional, and four for three-dimensional agents), with the distance to the closest intersecting object for each ray returned. In the case where the ray does

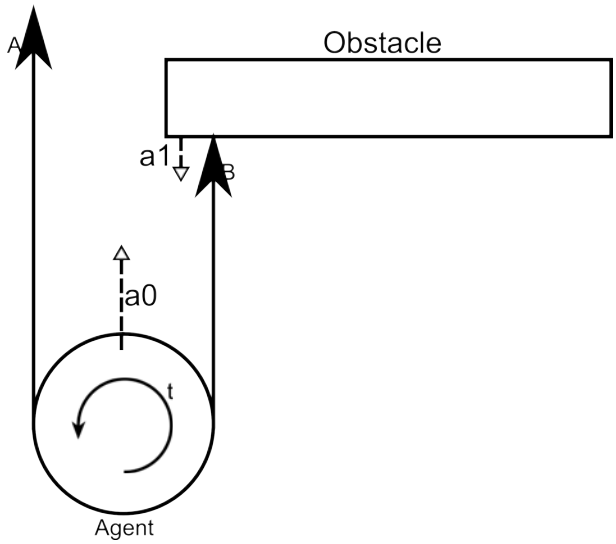

Figure 5.4: *The agent casts 2 rays, A and B, in its direction of orientation. These return the distance to the closest obstacle. Since A returns a larger value than B, the agent rotates towards that direction by applying a torque, t, to its current angular velocity. Additionally, the agent modifies its current acceleration a0 with an amount a1, so as to slow down sufficiently so that it stops before the obstacle should it be unable to steer clear of it.*

not intersect an obstacle, a value η, set to a pre-determined number which exceeds the dimensions of the scenario environment, is returned. If any of these distances are below some threshold, α, the agent enters collision avoidance mode, and the collision avoidance algorithm replaces the ANN as the agent controller.

$$\frac{-v^2}{2(d-\beta)} \tag{5.1}$$

The agent aims to steer in the direction of the ray with the largest distance value, and determines its acceleration using equation 5.1, where v is the current velocity, d is the smallest distance returned by a cast ray, and β is a threshold that indicates the closest possible distance between an agent and an obstacle (for example, if $\beta = 1$, the agent's velocity will have decelerated to 0 when it's 1 unit away from the obstacle). This is a reformulation of the kinematic equation $v_f^2 = v_i^2 + 2ad$, with $v_f = 0$ as we want the agents to come to a halt before they reach a distance β from the obstacle. A problem with the above equation is that in the rare case where $\beta > d$, the acceleration would end up being positive, resulting in the agent accelerating into the obstacle. To deal with this, we set $beta = 0$ if it is larger than d. Figure 5.4 illustrates this algorithm.

There are four cases where this algorithm fails to produce the desired results. The first is when other agents collide with the agent that is avoiding obstacles. This will often cause the agent to collide with the environment due to the extra force applied to it by other agents. We do not explicitly aim to resolve this issue as we aim to evolve inter-agent collision avoidance.

The second issue occurs when the orthogonal profile of an environmental obstacle is thinner than the agent. This may result in the agent not detecting the obstacle (fig-

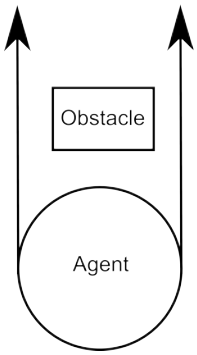

Figure 5.5: *Thin obstacles. An agent cannot detect an obstacle if it falls between the cast rays.*

ure 5.5), and thus, will not seek to avoid it. This is not a problem in our simulations as the obstacles are all larger than the agents. However, an easy method of dealing with this problem is to cast extra rays.

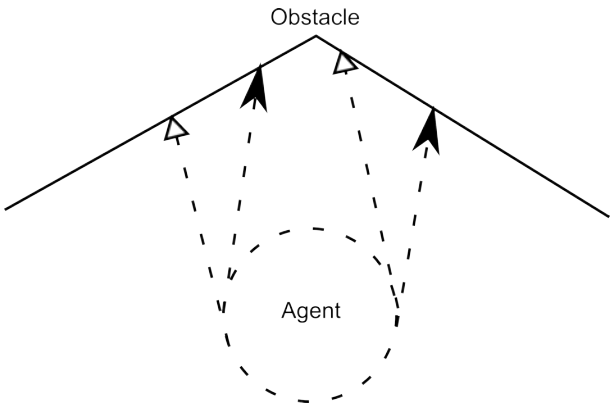

Figure 5.6: *Corner case example. The agent's current rays, denoted by the solid arrows, are longer on the left side than the right. This results in the agent rotating left. However, after rotating for a while, the right arrow becomes longer than the left (dotted arrows), resulting in the agent rotating back again, causing the agent to oscillate.*

The third issue occurs when an agent faces the inside of a corner. Rotating towards the ray with the larger collision distance results in that ray becoming shorter and the other becoming longer (figure 5.6), which in turn leads the agent to rotate back towards the original direction. This causes oscillating behaviour. In order to combat this problem, we first check if all the ray distances are below α, and if they are, we rotate towards the ray with the largest distance until one ray has a value larger than α.

The last issue with this algorithm occurs when two obstacles are separated by less than the width of an agent. Again this may result in oscillating movements. These oscillations occur because when the agent rotates towards the free ray, that ray encounters an obstacle and the previous ray is now freed (figure 6.1). This issue is resolved by having the collision avoidance algorithm ignore rays that are recently freed.

Our reasoning for using this rudimentary collision avoidance algorithm over a more robust and well established one [FS98, VdBLM08, OPOD10] is that it is very simple to

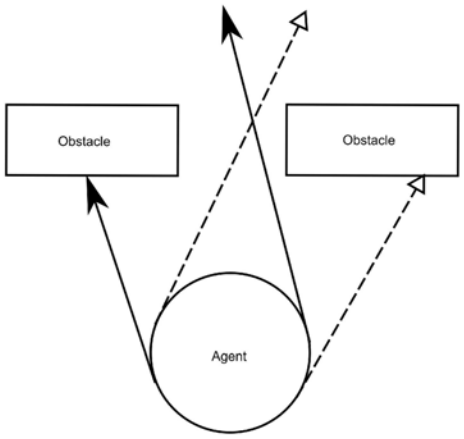

Figure 5.7: *Small gaps example. When one ray goes through the gap but the other hits an obstacle, it results in the agent rotating towards the direction of the gap. However, this results in a reversal of the situation, as shown by the dotted arrows. This causes oscillating behaviour for the agent.*

implement and integrate into our system with minimum additional computational cost, as we already employ ray-casting for other functions, such as agent vision (section 6.1). Additionally, our aim is not to evaluate the performance of various collision avoidance algorithms nor derive a new one, but rather to reduce the amount of learning required of the agents. Thus, an algorithm that produces adequate results is sufficient.

We found that using collision avoidance in our more complicated simulations allows for much easier learning. An example of this is the *Mouse Escape* simulation (section 6.2.7), where agents that were previously unable to evolve the desired behaviours are now capable of evolving them very quickly. This possibly shows that reducing the amount of control an ANN has over the agents can, in certain cases, speed up training and allow for more complex agent behaviours. This is interesting as it paves the way for possible future work, where agents can be modelled hierarchically [BG95, RMT01, PAB07], using ANNs for higher level functions, such as goals and psychological state, as opposed to directly controlling the low-level motor functions.

5.1.6 Distributing the System

The simulation sub-system is a bottleneck within our system. This is because each fitness evaluation requires the complete execution of a scenario, with thousands of fitness evaluations typically needed to complete the training process. This is problematic, since although the system does not have to be interactive, one would still ideally want reasonable training times. We use *under two hours* as a rough guideline for how fast our crowd simulations should evolve, as this is roughly the maximum amount of time users spent modifying very similar simulations in Jacka's [Jac09] work.

We aim to speed up the system by distributing it across multiple processes. This is achieved with a master-slave architecture, where the master node is responsible for running the evolutionary algorithm and assigning work to slaves, and the slaves are responsible for evaluating fitness. Algorithm 5 describes a slave process, and algorithm 6 details the fitness evaluation section of the master process.

Algorithm 5: Slave

```
Initialize simulation S;
while Termination message not received do
    Receive data d from master;
    if d is not termination token then
        Run simulation S with d;
        Acquire fitness f from S;
        Send f to master;
    else
        Terminate;
    end
end
```

Algorithm 6: Master

```
Initialize λ to number of fitness evaluations required this generation;
Initialize x to number of slave processes;
Assign chromosomes y_1..y_x to slaves slave_1..slave_x;
while There are unevaluated chromosomes do
    Find an unevaluated chromosome, y_k;
    Find a slave, slave_i, that has completed its work;
    Update the fitness received from slave_i to its respective chromosome;
    Assign y_k to slave_i;
end
Wait for slaves to finish work and update the respective fitnesses;
Terminate slaves;
```

The system automatically load balances, as a slave is assigned work when it completes its current job, resulting in faster slaves performing more work. The distributed computing framework used was Microsoft MPI[2] as it comes already compiled as part of the Microsoft HPC pack. However, our code adheres to the OpenMPI[3] standards, and thus can be compiled and run with other MPI frameworks.

5.2 Rendering Sub-system

The rendering sub-system is responsible for rendering a simulation scenario with a given candidate solution in real-time so that the behaviours of the agents can be visualized. It is also used for loading in assets such as 3D models and skeletal animations.

Rendering is accomplished by using the Ogre3D[4] graphics engine. To render a simulation, we read its current state, and then add, remove, or update the corresponding nodes in the Ogre scene graph. This scene graph is automatically rendered by Ogre each

[2]http://msdn.microsoft.com/en-us/library/bb524831(v=vs.85).aspx
[3]http://www.open-mpi.org/
[4]http://www.ogre3d.org/

frame. To run the simulation, we accumulate on each frame the time past since the last simulation cycle. Once this amount has reached the cycle time threshold (determined by the cycles per second attribute), we forward the simulation by one cycle and subtract the cycle time from the accumulated total. In the case that the accumulation is more than twice the cycle threshold, we forward the simulation by as many cycles as necessary.

In addition to rendering, we use Ogre to load assets such as models and animations, and to create the colliders for the agents and obstacles. This is required, as Ogre has its own format for 3D models and animations. One simplification we made when creating the colliders for the simulation rigid bodies is to use box colliders as opposed to arbitrary convex hulls, as we found that this allowed for significantly faster simulation with minimum impact on visual quality.

We chose to use Ogre as we found it easy to use, and because it is in the same language as our system (C++). It also has an active community, allowing for easier solution finding when encountering problems. Thus, if we encountered a problem, it was possible to easily find answers. Moreover, despite the fact that Ogre uses its own formats for 3D models and animations, a plug-in for Blender exists that allows one to export to them.

5.3 Summary

This chapter provides an overview on the workings of how our system simulates and renders crowds. It also gives information on how our system is distributed and how we performed inter-agent collision avoidance. Lastly, it provides some motivations for the libraries used. The full list of specific libraries, tools, and language used for our system can be found in appendix B.

Chapter 6

Scenarios and Agent Design

Crowd simulations are used in a variety of scenarios in films, from battles in *The Lord of the Rings* to swarming zombies in *World War Z*. Thus, in order to properly evaluate how well NE performs in controlling crowd behaviours, it is important to have a sufficiently diverse set of simulation scenarios. This chapter addresses the various scenarios and agents implemented in our system. These scenarios and agents are designed to be similar to the test cases found in Jacka's [Jac09] work as this allowed us to roughly compare both the behaviours and time taken.

6.1 Agents

There are five types of agents used in the scenarios: Car, Mouse, War robot, Space-ship, and Human. This section will describe the behaviours of each agent. The agent inputs, however, will not be discussed as they are simulation specific and will instead be discussed in section 6.2. An exception to this is agent vision, which is the same across all scenarios. We simulate vision by ray-casting, and return the distance to the closest intersected objects for each ray. An example of a mouse agent's vision is shown in figure 6.2.

The ANNs used for the agents are Feed Forward Artificial Neural-Networks with six hidden nodes, with the exception of NEAT which requires complexification (section 4.1.4). We found that this amount of hidden nodes worked well across all our agents using trial and error.

6.1.1 Car

The car agent moves in a two-dimensional plane. In order to control its behaviour, it receives two output values from the ANN. The first output modifies its acceleration, and the second applies a torque. In addition, the angular and linear velocity of the car agent is capped at a simulation specific amount, and the linear velocity of a car cannot be negative, thus not allowing the car agents to reverse. The no reversal rule is added in to make the objectives in the scenarios more challenging for the agents, as they are then forced to change their front-facing orientation rather than just learning to reverse.

The vision of a car agent is simulated by casting eight rays uniformly around it. The motivation for this is that cars have rear and side-view mirrors, allowing the drivers to see in all directions.

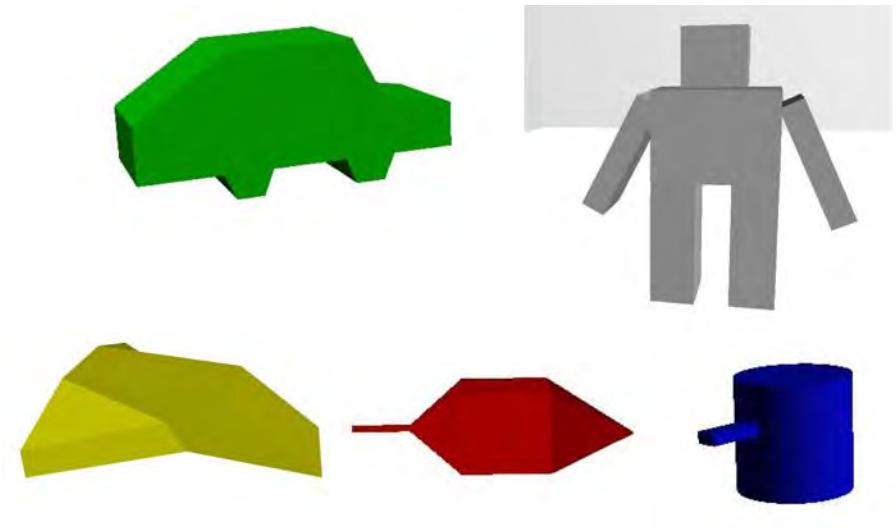

Figure 6.1: *The five different agent types we use for our simulation scenarios. From left to right and top to bottom: Car, Human, Space Ship, Mouse, and War-Robot.*

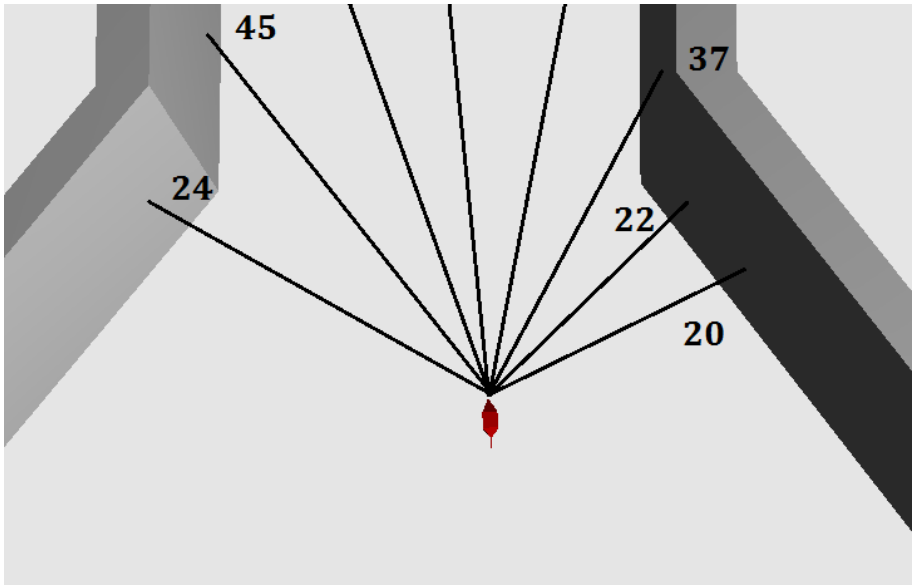

Figure 6.2: *An agent's vision is simulated by casting rays, and returning the distance to the closest intersection for each ray. This figure shows how we simulate the vision of a mouse agent.*

6.1.2 Mouse

Much like the car agent, the mouse also moves in a two dimensional plane with the ANN determining both torque and changes in acceleration, with both the linear and angular velocity capped at a scenario specific amount and the linear velocity not allowed to be negative. The mouse agent also has the additional behaviour that it can sense obstacles in front of it, and if they are too close, it will come to a halt. The vision of a mouse agent is simulated by casting eight rays in a cone in front of it. The reason for this is that a front-facing cone-like field of view is common for many living organisms, so we made the assumption that a mouse's field of view is similar.

6.1.3 War Robot

The war robot agent has identical movement rules to the mouse and car agent, with the exception that its angular velocity is fixed at zero after its linear velocity exceeds a specific threshold. This is inspired by heavily armoured vehicles, where turning at high velocities is dangerous due to the momentum and lack of grip.

The vision of the war robot agent is similar to that of the car agent, with eight rays cast uniformly around the agent. However, in addition to the distances provided, the vision is also capable of determining if the object intersected is an enemy, friendly, or an environmental obstacle. If the object is an enemy, is within firing range, and within the line of fire, the robot will fire at it and subsequently destroy it. Further, the fire action has a cool-down timer. Thus, the robot is unable to continuously fire and destroy multiple enemies in quick succession. This firing action was designed to be independent from the ANN outputs as we found that it was very difficult to train the ANN to perform this action together with movement.

6.1.4 Human

The human agent's movement rules are identical to that of the car and mouse agents, and like the mouse agent, it is able to detect if there is an obstacle in front of it and immediately halt if the obstacle is close enough. The ANN also has a third output value which, when above a certain threshold, will cause the human agent to exert a push action upon another human agent directly in front of it (within one unit). When pushed, an agent will stumble and get trampled, resulting in it being removed from the simulation. This is to allow for frantic evacuation or chase scenarios, such as scenes in *World War Z* or *Pompeii*, where agents often stumble and trample over each other in order to escape some hazard or reach some goal. The vision of a human agent is simulated by casting eight rays in a cone in front of it.

6.1.5 Spaceship

The spaceship agent moves in three-dimensional space. The ANN provides three output values to this agent, modifying its acceleration, and applying torque along the x and y axes. As with previous agents, its velocity is capped and is non-negative. The ANN does not apply torque along the z axis as we found that this resulted in an undesirable spiralling behaviour. The vision for this agent is simulated by casting rays in a 5x5

square grid in front of the agent, which simulates a field of view in three dimensions. We decided against casting rays uniformly around the agent as we observed that the agents did not behave significantly differently. Additionally, uniformly casting rays around the agent requires many more rays, leading to an increase in computational complexity and problem dimensionality.

6.2 Scenarios

We use eleven crowd simulation scenarios as our test cases. Although these scenarios do not cover all the use-cases of controlling crowds, they do represent sufficient variety for testing the validity of NE for controlling crowds.

Throughout the various simulation scenarios, we make use of a set of recurring objectives:

- C - The collision objective, where agents are tasked to avoid collisions with one another. Every simulation cycle, each agent's distance to its neighbouring obstacles and agents is calculated. If the distance is below α, a value between zero and one is added to the total, with zero chosen if the distance is α, and one chosen if the agent and the obstacle are colliding. We set α to five units for all our simulations. This value was obtained through experimentation.

- D - The goal objective, where agents are tasked to either move to a specific point, or to cross a finishing line before the end of the simulation. This value is the sum of the distances of all the agents to the goal point or line at the end of the simulation. Only agents which have yet to reach the goal are considered.

- P - The group size objective, where the simulation is tasked to have a set amount of agents left within a group at the end of the simulation. The value of this objective is calculated by getting the absolute value of the difference between the actual and desired agent group sizes. One problem with this objective is that its value is very small compared to the other objectives, which may lead to the crowd not learning this objective. To combat this, we assign very large weights to this objective.

- A - The angular velocity minimization objective, where agents are tasked to keep their angular velocity as low as possible. One problem we found with our simulations is that agents often learned an oscillating behaviour when moving from one point to another. This is undesirable as one tends to minimize the amount of turning when travelling to a specific location. We use this objective to eliminate this oscillating behaviour, where the sum of the angular velocities of every agent is added to the total during each simulation cycle.

In addition to the above four objectives, we also make use of two more simulation specific objectives, which we will further elaborate on in their respective sections.

We also define a fitness threshold for each of the simulations. This threshold signifies the fitness that a candidate solution needs to achieve in order to sufficiently adhere to the specified objectives. These thresholds were determined through preliminary experimentation.

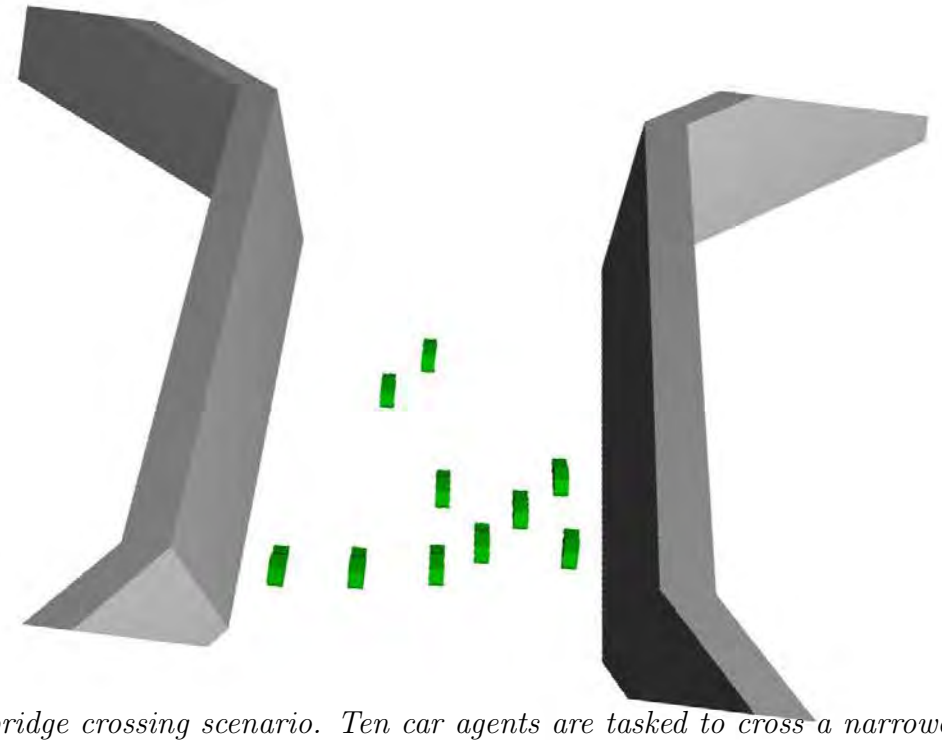

Figure 6.3: *Car bridge crossing scenario. Ten car agents are tasked to cross a narrower bridge area before a time limit of ten seconds.*

6.2.1 Car Bridge Crossing

Ten car agents starting in a wide area are tasked to cross a narrower bridge area within ten seconds whilst avoiding collisions with both the environment and other agents (figure 6.3). This scenario tests the agents ability to navigate through a bottleneck at a considerable pace, while avoiding collisions. Due to the simplicity of the scenario, the environment collision avoidance algorithm is not used for this scenario.

The car agents, in addition to having vision as inputs, also provide the ANN with their current position, the line at the end of the bridge that the agent must cross, their current linear velocity, and their current angular velocity. Additionally, the car agents' linear velocity is capped at ten, and their angular velocity is capped at one.

The fitness of a chromosome is calculated using $f = 2D + 2C + A$, and the fitness threshold is 100.

6.2.2 Mouse Bridge Crossing

This scenario is identical to the Car Bridge Crossing scenario, with the exception that instead of ten car agents, 30 mouse agents are simulated instead (figure 6.4). As there are more agents, and due to the halting behaviour of the mice agents, crossing the bridge within the specified time limit becomes significantly more difficult. This results in the agents having to learn more efficient behaviours in order to succeed. The fitness threshold for this scenario is defined at 750.

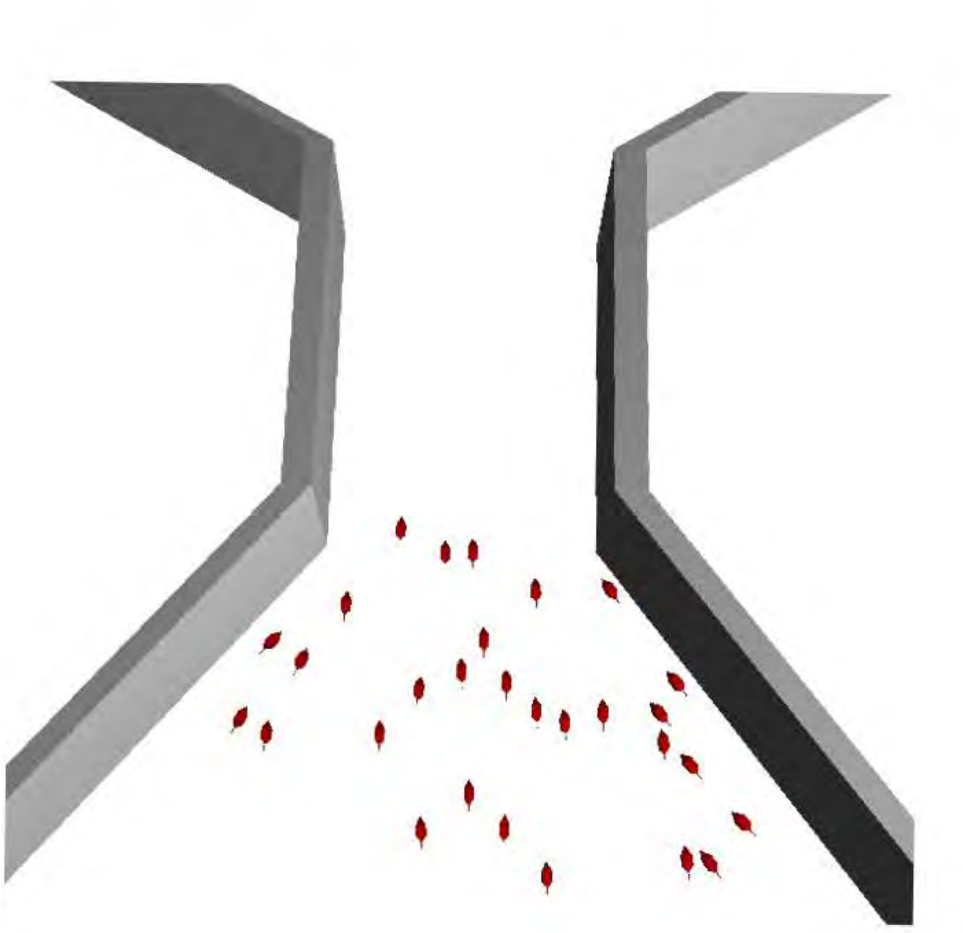

Figure 6.4: *Mouse bridge crossing scenario. Thirty mouse agents are provided goals to cross the narrower bridge area within ten seconds.*

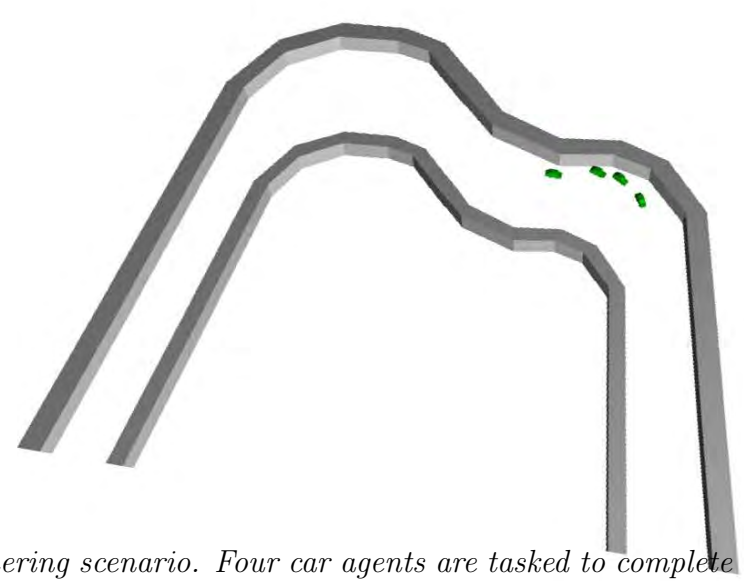

Figure 6.5: *Car cornering scenario. Four car agents are tasked to complete the race track in 13 seconds.*

6.2.3 Cornering

Four car agents are tasked to complete a winding race track in 13 seconds while avoiding collisions (figure 6.5). This scenario tests the ability of agents to navigate around a track in a specific amount of time whilst avoiding crashes on the track corners. No agent to environment collision avoidance was used as due to both the simplicity of this scenario, as well as avoiding collisions being one of the main objectives of this scenario.

One problem we found with the training is that it was difficult for the agents to learn how to go to the final goal due to the winding track. In order to alleviate this problem, we used goal points. These are points placed at every corner, as well as at the end goal. The initial agent goal is then assigned to the first goal point, with the goal being updated to the next goal point every time an agent reaches its current goal.

The additional agent inputs for this scenario are its linear velocity, its current goal point, its position, and its angular velocity. The agents' linear velocity is capped at 15, and its angular velocity is capped at one.

The fitness obtained by a chromosome can be obtained by using $f = C + 2D + A$, and the fitness threshold is defined at 300.

6.2.4 Car Race

Ten car agents are tasked to race across a track within 13 seconds while avoiding collisions, with the agent at the back tasked to win the race (figure 6.6). This scenario is similar to scenes in various racing films, where an underdog comes from behind to win a race. The environment only collision avoidance algorithm was used for this scenario.

The car agents are provided additional inputs in the form of position, the points defining the finishing line, velocity, the underdog's position, and a flag specifying whether or not they are the underdog destined to win the race. The agents' linear velocity is capped at 15 and their angular velocity as one. Additionally, the agents are given an initial velocity of two. This is to prevent cases where the agents learn to not accelerate

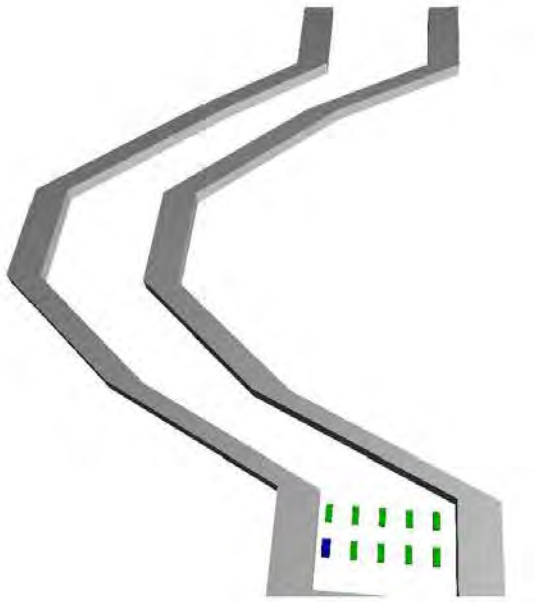

Figure 6.6: *Car race scenario. All cars must finish the race-track within 13 seconds, with the blue car finishing first.*

in the first few seconds if they are not the underdog, which leads to an unrealistic scene as it would result in the competitors cooperating.

The fitness for this scenario is calculated using $f = C + D + 20W$ where W is the distance of the underdog to the actual winner of the race at the time the first car crosses the finish line. The fitness threshold of this scenario is 250.

6.2.5 Car Crash

Two groups of ten car agents are initialized opposite to each other in a narrow corridor, with the aim being that they need to move to the opposite end of the corridor within ten seconds while avoiding collisions between both the environment and other cars (figure 6.7). This scenario is analogous to various scenes in action movies, where cars or jets move towards each other but skilfully maneuver to avoid collisions. As the primary aim of this scenario is to have the agents learn to avoid collisions against oncoming traffic in a tightly packed space, collision avoidance is not used.

Inputs for the car agent in addition to its vision include its position, the two points depicting the finish line of its respective group, and its velocity. The linear velocity is capped at 15, and angular velocity capped at one.

The fitness is calculated with $f = C + 5D$, and the fitness threshold is 250.

6.2.6 War Robot Battle

Two group of war robot agents are tasked to battle each other for ten seconds. One group is initialized behind various buildings, whereas the other is initialized in an open field. The aim of this scenario is to have between nine and 11 agents left in each group at the end of the scenario (figure 6.7). This scenario tests how capable NE is at scenarios where battle outcomes need to be controlled. Collision avoidance was used for this scenario.

Figure 6.7: *Car crash scenario. Cars must avoid collisions with other cars and move to the opposite end of the corridor within ten seconds.*

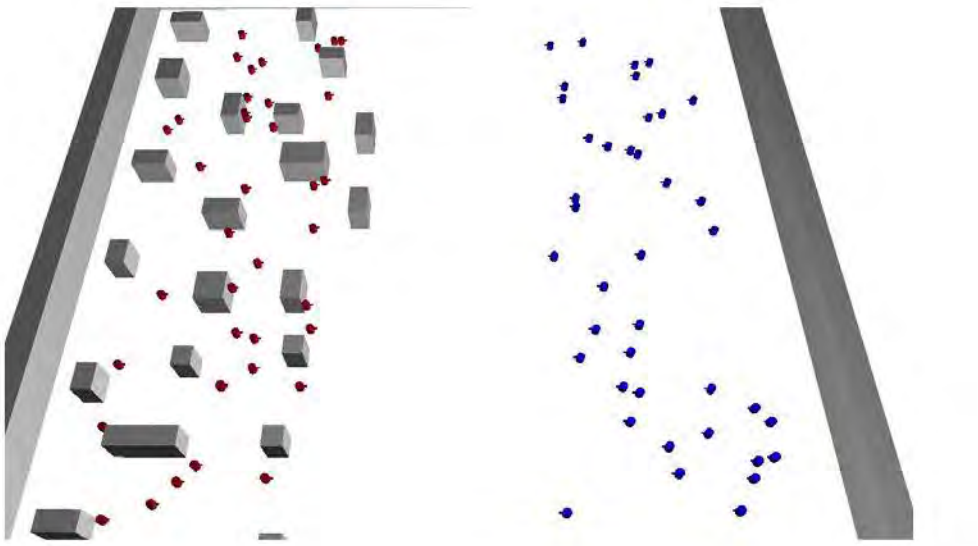

Figure 6.8: *War robot battle scenario. Eighty agents are separated into two groups of 40, and are tasked to battle each other. The aim is that there are nine to 11 agents left on each side after ten seconds of simulation.*

The additional inputs for the war robot agents are agent position and agent velocity. The linear velocity is capped at 15, the angular velocity is capped at one, and the shooting cooldown is set at 0.5 seconds.

The fitness for this scenario is calculated using $f = C+20P$, with the fitness threshold being defined at 150.

6.2.7 Mouse Escape

Thirty-five mouse agents are tasked to escape from five robot agents. The mouse agents are initialized on one side of the environment, behind various buildings which act as cover, and have to cross a finish line at the other end of the environment in order to count as having escaped (figure 6.9). The robot agents are initialized opposite to the mouse agents. The scenario's goal is for there to be 11 to 13 mouse agents having escaped at the end of a 15 second simulation, with the rest of the mouse agents being shot and killed by the robots. Collision avoidance was used for this scenario.

The mouse agents have the additional input of their position, linear velocity, and the points depicting the finish line. The war robot agents have additional inputs of their position and velocity. The linear and angular velocities of the mouse agents are capped at 15 and one respectively, whereas the war robot agents' were capped to ten and one.

The fitness of the scenario is calculated by using $f = 4D + 25P + C$, and the fitness threshold is set at 100.

6.2.8 Human Evacuation

Sixty human agents were tasked to evacuate a room in 11.5 seconds, with the aim of there being 13 to 15 agents being alive and having evacuated by the end of the simulation, with the rest being trampled to death (figure 6.10). This scenario is similar to the mouse escape

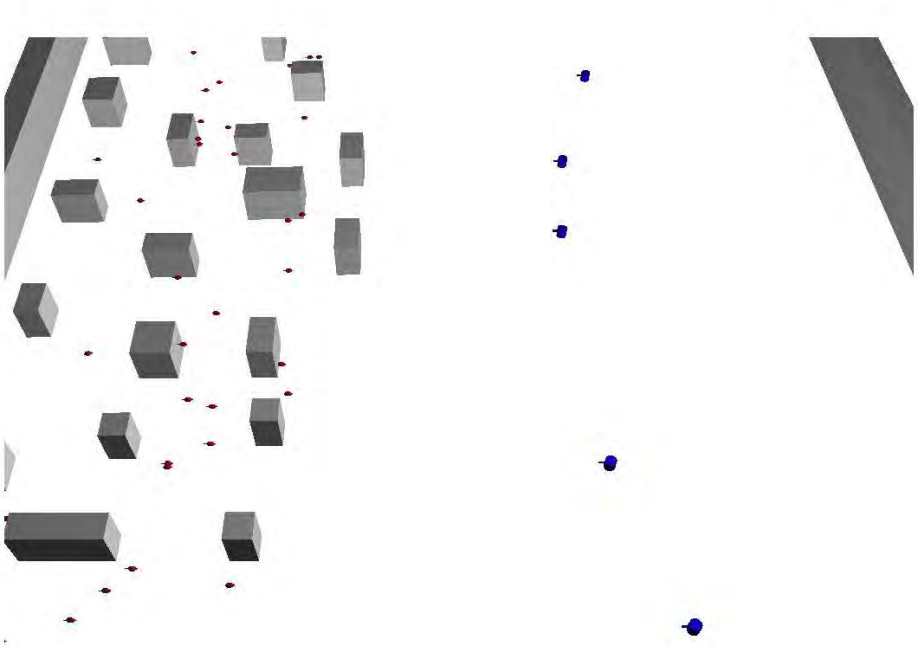

Figure 6.9: *Mouse escape scenario. Thirty-five mice tasked to escape five war-robots. Eleven to 13 mice must be left alive at the end of a 15 second simulation. All of these mice must also have escaped (reached the opposite end of the environment).*

scenario in that it tests how well NE performs in controlling how many agents survive when escaping from some hazard. However, it differs in that the hazard is not an external factor but rather internal to the crowd.

In addition to the human agent's vision, they are also provided with their current position, their current velocity, their current angular velocity, and the points denoting line segment representing the exit of the room. The human agents' linear and angular velocities were capped to 15 and two. Collision avoidance is used for this scenario.

The fitness of a chromosome in this scenario is calculated using $f = D + 60P + E$ where E is the sum of the distances from where each agent dies to the exit. We used E as we found that often the agents would prioritize trampling over evacuating, as it lead to more efficient movement. This caused most agents to get trampled very far away from the exit, thus the bottleneck trampling behaviour we desired was not learned. The fitness threshold for this scenario is set at 1000.

6.2.9 Spaceship Turn Back

Forty spaceship agents are tasked to move towards a goal sphere within ten seconds. This goal sphere is initialized behind the agents' starting positions (figure 6.11). This scenario tests the ability of NE to control three dimensional behaviour of agents. Additionally, an extra goal sphere with a significantly larger radius is used to ensure cohesion amongst the crowd of agents. Although it is also possible to use the sum of the distances to the crowd centroid as an objective to achieve this, we found using extra goal spheres to be a much simpler method. Collision avoidance was not used for this scenario.

The spaceship agents are provided the additional inputs in the form of current agent position, position of the goal sphere, current linear velocity, and the magnitude of the an-

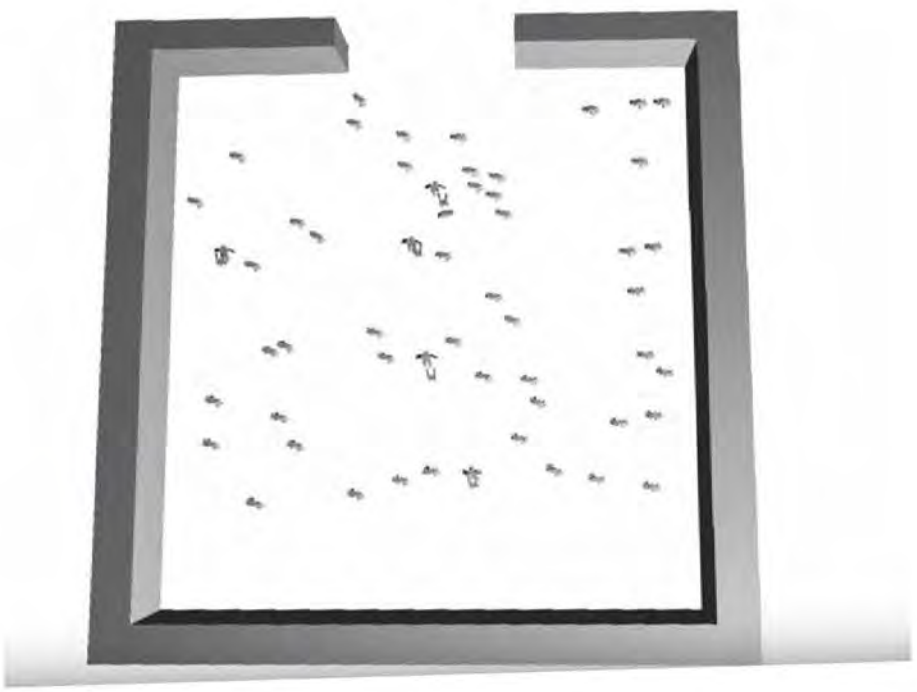

Figure 6.10: *Human evacuation scenario. Thirteen to 15 agents tasked to evacuate room in 11.5 seconds, with the rest being trampled*

gular velocity. The agents linear velocities were capped at 25, and their angular velocities capped at two.

The fitness for this scenario is calculated using $f = 10C + 2G_1 + 2G_2 + 0.2A$ where G_1 is the goal sphere, G_2 is the extra goal sphere used to enforce cohesion. The fitness threshold is set at 750 for this scenario.

6.2.10 Spaceship Obstacle

This scenario is identical to the spaceship turn back scenario, with the exception of the goal spheres and the environment. The goal spheres are placed in front of the agent, with a large obstacle in between the goal point and the agent starting positions (figure 6.12). The goal of this scenario is to test if agents can be evolved to navigate around a large obstacle in order to reach their goal. The fitness threshold is set at 1000 for this scenario.

6.2.11 Spaceship Obstacle Field

This scenario is identical to the spaceship obstacle scenario, with the exception of there being a field of small obstacles instead of one large one (figure 6.13). This tests the agents ability to learn to avoid these small obstacles when moving towards the goal. The fitness threshold, for this scenario, is set at 1000.

Figure 6.11: *Spaceship turn back scenario. Forty spaceship agents tasked to move towards the goal point denoted by the blue circle.*

Figure 6.12: *Spaceship obstacle scenario. Forty spaceship agents assigned the task of navigating around the obstacle in order to reach the goal point denoted by the blue circle.*

Figure 6.13: *Spaceship asteroid field scenario. Forty spaceship agents must navigated through the asteroid field in order to reach the goal point, denoted by the blue circle.*

6.3 Summary

This chapter provides details on the design of both our agents and simulation scenarios. Information such as the agents' perceptual abilities, behaviors, and constraints, as well as scenario objectives and fitness calculations are provided. These scenarios and agents are designed to be similar to the ones found in Jacka's [Jac09] work, so that his data can be used as a rough guideline for determining if our system performs acceptably.

Chapter 7

Experimental Setup

It is important to design our experiments so that the data obtained from them can be used to answer our research questions (section 1.1). This chapter will elaborate on these experiments, and draw connections to which research question each aims to answer. This chapter also details what NE parameters and operators are used, and provides motivation as to why we chose them. This is necessary because the performance of the NE algorithms are highly dependent on the specific parameters used, with the performance of these parameters in turn being problem dependent.

7.1 Experiments

This section aims to elaborate on our various experiments. We also provide details on the general experimental parameters (non-NE), as well as motivation for our choice in the type of statistical test used.

7.1.1 General Experimental Parameters

The experiments consist of a set of algorithms or parameters that are compared against each other, using fitness as the performance metric. In order to accomplish this, we run the evolutionary phase 30 times (general rule of thumb for a sufficient sample size [HTR93]) for each algorithm or parameter, sampling the fitness values of the best performing chromosomes within the algorithm every 10 fitness evaluations to provide sufficiently high resolution data on how well the algorithm performed throughout the course of the evolutionary process. Thus, $n/10$ populations of 30 data points are generated for each test case (required because NE algorithms are stochastic), where n is the total number of fitness evaluations during evolution. These populations can then be compared with corresponding populations from a different test case in order to determine the best performers at various stages of evolution.

We chose 20,000 fitness evaluations as the cut-off for most of our experiments. The reasons for this are two-fold: Our preliminary experiments showed that there is little improvement after this, as the gradients of the mean fitness zero out around this point, and we found that this number of fitness evaluations allowed for evolution times similar to those obtained by Jacka [Jac09] for similar scenarios.

An important aspect of activation functions is their *active range*, which specifies the range in which the gradient of the function is not close to zero. This indicates what range the ANN inputs should fall in order to achieve effective evolution. The active range for the activation function used in our system, the *Sigmoid*, is $(-\sqrt{3}; \sqrt{3})$ [Eng07]. For this reason, we normalize the inputs to this range, and restrict the weights to be between the range $[-1; 1]$. An exception to this is CMA-ES, where we have not limited the weights. The reason for this is because clipping or wrapping weights will result in the distribution being incorrect, significantly hampering the evolutionary process. Instead, we initialize the algorithm with a small initial step size, which we found was sufficient to keep the weights small.

The statistical test used to test for significance between the results is the one-tailed Mann Whitney U test [MW+47]. This test was chosen because it is robust to non-normally distributed populations, an attribute that results in our preliminary experiments often exhibited. We test for significance by running this test between the best performing test case and all other test cases. We regard $p < 0.05$ as being statistically significant. As the objectives in our scenarios aim to evolve agents to reduce some behaviour (for example reduce distance to goal and reduce number of collisions), better performing controllers will receive lower fitness values. Thus, in our results, the best performing test case will be the one that has the lowest mean fitness value.

7.1.2 Determining the Best Performing Algorithm

One of the main research goals for this thesis is to determine the best performing NE algorithm for controlling emergent crowd behaviours. We achieve this by comparing the fitnesses obtained by the four implemented NE algorithms (section 4.1) across our 11 different simulation scenarios (section 6.2).

7.1.3 Determining the Best Team Architecture

As discussed in section 3.3.5, there are various agent controller team compositions, ranging from fully homogeneous to fully heterogeneous. However, there is no real indication as to which compositions work well in the context of controlling crowd simulations. Thus, an additional research goal of this thesis is to investigate the impact different team compositions have on the evolutionary process.

Another consideration is how these various team compositions affect the behaviour of the agents. This is important to investigate as our preliminary experiments, which used fully homogeneous controller setups, showed that the agents' behaviours were overly homogeneous in certain scenarios, which leads to a lack of perceived realism.

To investigate these issues, we have created four different controller setups for each simulation scenario, and evaluate both the fitnesses obtained and the heterogeneity of the behaviour. Specifically, with regards to the heterogeneity of the behaviours, we are interested in the *believable heterogeneity* of agents, meaning that they should not deviate in their behaviours simply for the sake of deviation, but should rather achieve their goals using a variety of believable strategies. As performing user evaluations falls outside the scope of this project, we will instead evaluate this aspect by rating how efficient and homogeneous the crowds appear to behave. The heterogeneous rating determines how

differently the agents behave, whereas the efficiency rating determines how believably they behave. This is based on the *principal of least effort* [Zip49], which states that humans look to minimize the amount of effort spent in accomplishing their goals. These ratings will be obtained by having 10 people in our lab conduct an informal user survey[1], where they have to rate videos of each test case between one to five for both how efficient, and how heterogeneous they behave. As the sample size is small, these results will only serve to provide us a general idea, with a larger population size with a better designed test being needed for a clearer picture of the believability of the various team compositions. The NE algorithm we use will be the best performing algorithm found in section 8.1.

As each simulation scenario has different numbers of agents, the four different controller setups for our simulations also need to be scenario specific. The controller setups used for our simulation scenarios are as follows:

- **Car Bridge Crossing (10 agents)** - one ANN per agent (Het), one ANN per two agents (SHet), one ANN per five agents (SHom), one ANN for all agents (Hom).
- **Mouse Bridge Crossing (30 agents)** - same as Car Bridge Crossing.
- **Cornering (four agents)** - one ANN per agent (Het), one ANN per two agents (SHet), one ANN for all agents (Hom). Only three setups are required as there are only four agents in the scenario.
- **Car Crash (two groups of 10 agents each)** - one ANN per agent (Het), one ANN per five agents (SHet), one ANN per group (SHom), one ANN for all agents (Hom).
- **Car Race (10 agents)** - one ANN per agent (Het), one ANN per two agents (SHet), one ANN for the underdog and one ANN for the rest of the agents (SHom), one ANN for all agents (Hom).
- **War Robot Battle (two groups of 40 agents each)** - one ANN per agent (Het), one ANN per four agents (SHet), one ANN per 10 agents (SHom), one ANN per group (Hom).
- **Mouse Escape (two groups, one of 35 mouse agents, one of five robot agents)** - one ANN per mouse agent (Het), one ANN per five mouse agents (SHet), one ANN per seven mouse agents (SHom), one ANN for all mouse agents (Hom). one ANN is used for all five robot agents for all cases, as we found them to be somewhat difficult to divide up.
- **Human Evacuation (60 agents)** - one ANN per agent (Het), one ANN per five agents (SHet), one ANN per 10 agents (SHom), one ANN for all agents (Hom).
- **Spaceship Turn-back (40 agents)** - one ANN per agent (Het), one ANN per four agents (SHet), one ANN per 10 agents (SHom), one ANN for all agents (Hom).
- **Spaceship Obstacle** - Same as Spaceship Turn-back.

[1] https://docs.google.com/forms/d/163-zwjYZUCqZPpIvLv7eK4BHggprJNXx8Up1Ja4P7-o/viewform?usp=send_form#start=invite

- **Spaceship Obstacle Field** - Same as Spaceship Turn-back.

In the results chapter, we will refer to these compositions as Heterogeneous, Semi-Heterogeneous, Semi-Homogeneous, and Homogeneous. The difference between Semi-Homogeneous and Semi-Heterogeneous team compositions is that Semi-Homogeneous has fewer and larger groups, and Semi-Heterogeneous has more but smaller groups. These are denoted as Het, SHet, SHom, and Hom in parentheses in the list.

7.1.4 Determining Feasibility

In order to test whether it is possible to use NE to control emergent crowd behaviour, and the feasibility of doing so in films, we run the best training algorithm, together with the best respective team architecture across all 11 scenarios for 20,000 fitness evaluations.

In the case that the fitness of a candidate solution reaches the acceptable fitness threshold for a scenario, evolution is terminated and the best candidate solution is returned. The evolutionary stages that terminate before 20,000 fitness evaluations will be marked as successes, whereas the ones that reach 20,000 fitness evaluations will be marked as failures. This will provide a success rate, which will help us to determine whether or not it is possible to control emergent crowd behaviours using NE. In order to test if the evolution times are feasible for use in films, we record and evaluate the average evolution times. We will compare these evolution times with the time measurements from Jacka's work [Jac09] as a rough guideline in order to determine if they are feasible in practice.

7.1.5 Evaluating Scalability

As the system is distributed, it is useful to see how well it scales when more processes are added. Although not an explicit research question, it is nonetheless important to evaluate this, as a better scaling system will allow for the evolution of larger and more complex crowds within the same amount of time. For the purposes of this test, we will compare the execution times of the best performing NE algorithm when run for 20,000 fitness evaluations when using one, two, three, and four slave processes across all 11 scenarios.

7.2 Parameter and Operator Setup

The parameters and operators in an NE algorithm play a pivotal role in determining how well it performs. As these are often task dependent, it is important to select them correctly. This section aims to motivate and detail the parameters and operators selected that had the most impact on the evolutionary process, providing either statistics in the case of the more important operators, or through observations obtained via trial and error. The scenarios for these preliminary tests were chosen to cover a range of different scenarios, so as to provide a better idea of how the tested parameters and operators generalize. The rest of the parameters used can be found in appendix C. In the result tables of this section (and the results chapter), we abbreviate the scenarios to the first letters of their names. For example, Mouse Bridge Crossing is abbreviated to MBC, and Car Race is abbreviated to CR.

7.2.1 NEAT

The parameters and operators used for NEAT are mostly the default recommended values [SM04], with the exception of the population size, elitism, compatibility threshold, mutation operator, and selection operator, as we found that these significantly impacted the evolutionary process.

A population size of 100 is used as we found that it provides a good balance between consistency and computational cost, and elitism of 5% is chosen as we found through experimentation that it provided a good selective pressure to speed up evolution, without causing much premature convergence. We also found that the truncation selection used in the original algorithm [SM04] results in premature convergence. Thus, we instead look to use Linear Rank-Based selection. The reasons for selecting this specific operator are discussed in section 7.2.5. We also use Gaussian mutation with a standard deviation of 0.2 to perturb the weights as opposed to Uniform mutation. The reasons for this are discussed in section 7.2.7

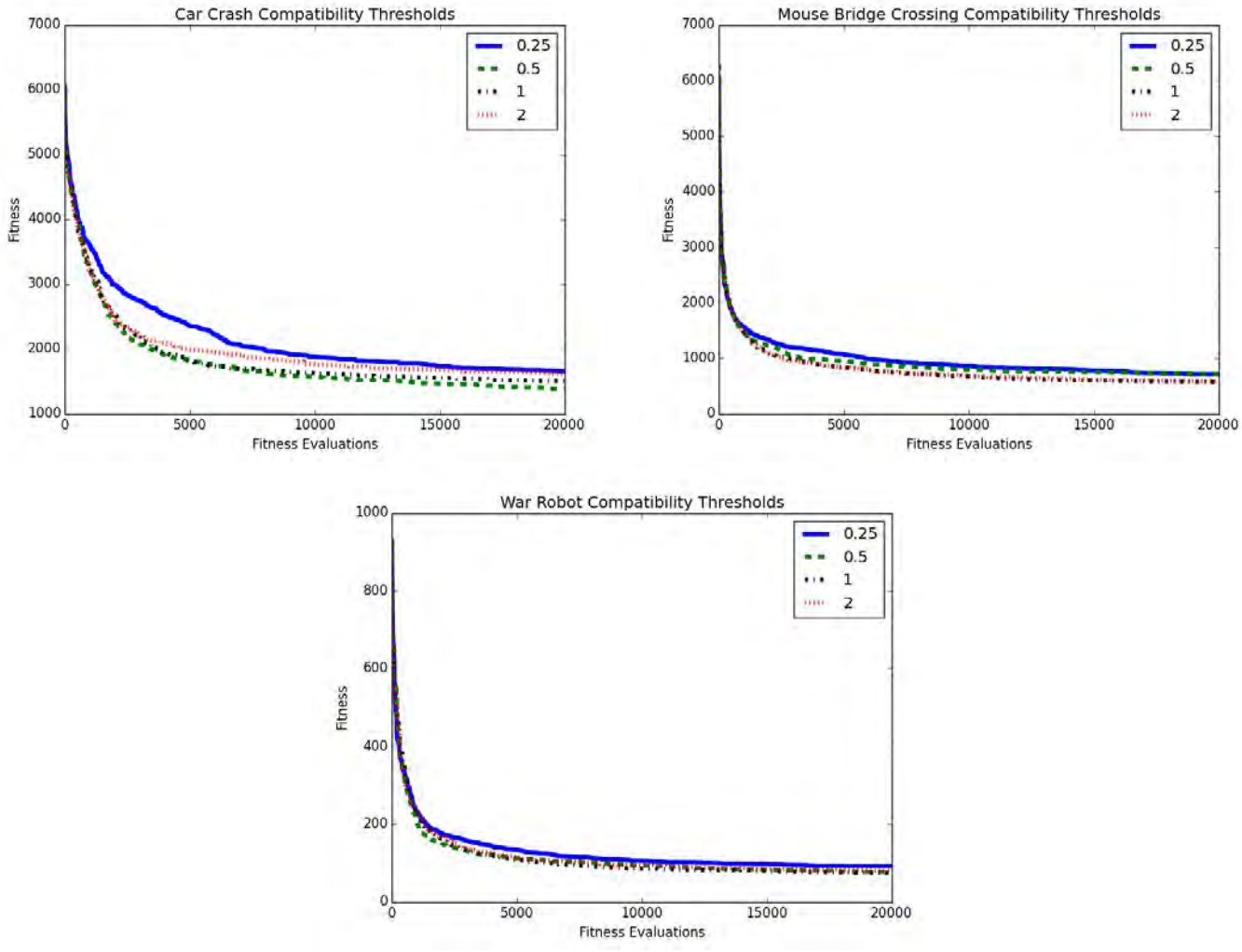

Figure 7.1: *Performance of different compatibility thresholds over 20,000 fitness evaluations in the Car Crash, Mouse Bridge Crossing, and War Robot Battle scenarios.*

Table 7.1: *Mean fitnesses and standard deviations achieved by each compatibility threshold after 20,000 fitness evaluations, with bold values being below the fitness threshold of the respective scenario. P-values obtained using the Mann-Whitney U test for are given in parenthesis. These p-values are obtained by comparing the current compatibility threshold with the compatibility threshold that achieved the lowest mean. A dash is given instead of the p-value if the corresponding compatibility threshold achieved the best mean for the specific scenario.*

Scenario	**0.25**	**0.5**	**1**	**2**
MBC	712.36 (2.9e-3)	715.5 (1.9e-3)	**576.53 (-)**	582.98 (0.5)
MBC σ	131.81	152.79	214.64	139.14
CC	1659.24 (0.02)	**1385.59 (-)**	1503.92 (0.5)	1627.79 (0.054)
CC σ	551.03	362.42	678.56	657.76
WRB	91.7 (5.5e-7)	76.69 (0.28)	**74.13 (-)**	79.41 (0.13)
WRB σ	9.2	15.08	13.07	14.92

We also performed a series of preliminary tests to determine the ideal Compatibility Threshold. These tests were run because we found that the recommended threshold of 3 [SM02] did not allow our ANNs to speciate. As a consequence, we tested the compatibility thresholds at 0.25, 0.5, 1, and 2 across three representative scenarios in order to find the better performing threshold for us to use in our experiments. Figure 7.1 and table 7.2.1 details our findings. Overall, we noticed that thresholds of both 1 and 2 performed consistently well. However, as a fitness threshold equal to 1 always achieved lower means than a fitness threshold of 2 (despite the two being statistically comparable for all the scenarios), we decided to use it as the final compatibility threshold.

7.2.2 CNE

We decided to use a population of 100 for CNE as, much like NEAT, we found that it provided a good balance between consistency and computational cost. We also use elitism of 5%, as we found that the selective pressure provided greatly helped the speed of evolution, without deteriorating the evolutionary process much with premature convergence. The rest of the operators and parameters used for CNE are discussed in sections 7.2.5 - 7.2.7.

7.2.3 ESP

For ESP, we set the sub-population size and number of fitness evaluations per neuron to 50 and 5 respectively, as we found that they provide a balance between consistency and number of generations run (as we stop the training after 20,000 fitness evaluations). Elitism of 10% is used in order to speed up the evolutionary process. This higher elitism percentage could be used in this algorithm as burst mutation helps alleviate premature convergence.

A stagnation threshold of 20 generations is used to detect whether a sub-population requires burst mutation. We found that values lower than 20 resulted in running burst

mutation on populations that have not yet converged, whereas values larger than 20 leads to slow reaction times, allowing a population to stay converged for much of the training process.

For burst mutation, we sample values around a Gaussian distribution with mean 0 and standard deviation of 0.2, as we found in section 7.2.7 that this is the best mutation configuration.

As with CNE, the mutation, crossover, and selection operators and parameters are discussed in sections 7.2.5 - 7.2.7.

7.2.4 CMA-ES

One of the strengths of CMA-ES is that almost all of the parameters and operators are self-adapting. One parameter, however, that has not been defined is the initial step size. We ideally want a small step size as it helps keeps the ANN weights small. We determined that an initial step-size of 0.2 is sufficient for this through experimentation.

7.2.5 Selection Operator for NEAT, CNE, and ESP

As discussed in section 4.4, we have implemented four different selection operators in our NE sub-system. In order to determine which operator to use in our experiments, we obtained data for all four across three scenarios: Car Crash, Mouse Bridge Crossing, and War Robot Battle. Although it is ideal to test these operators with NEAT, CNE, and ESP, we decided to only test them with CNE due to time and scoping constraints.

As shown in figure 7.2 and table 7.2.5, Linear Rank-based selection performed the best, achieving the best mean fitness in two of the three scenarios, and a mean fitness comparable to the best in the third. For this reason, we have decided to use Linear Rank-based selection over Boltzmann for our final experiments.

We also found that, despite Non-linear Rank-based selection performing very poorly, its initial improvement is the fastest. This is likely because it causes too much selective pressure, resulting in premature convergence. Thus, it may be possible to use various re-initialization strategies or a different non-linear function that provides less selective pressure in order to improve its performance.

7.2.6 Crossover Operator for CNE and ESP

Genome crossover is another important operator that determines the performance of an NE algorithm. In order to evaluate the best performing operator implemented, we compared them across 5 different crossover probabilities (ranging from 0.2 to 1) in 3 representative simulation scenarios. The algorithm used with these two operators was ESP, and the scenarios used are Cornering, War Robot Battle, and Mouse Bridge Crossing. ESP was used for this study as the results were obtained through an independant study from the results for both mutation and selection.

Table 7.2.6 shows some of the best and worst performing operators in our experiments, and figure 7.3 shows the mean fitness obtained by each operator after 200 generations in ESP. Full results can be found in our online repository[2].

[2]https://bitbucket.org/igorawratu/thesis

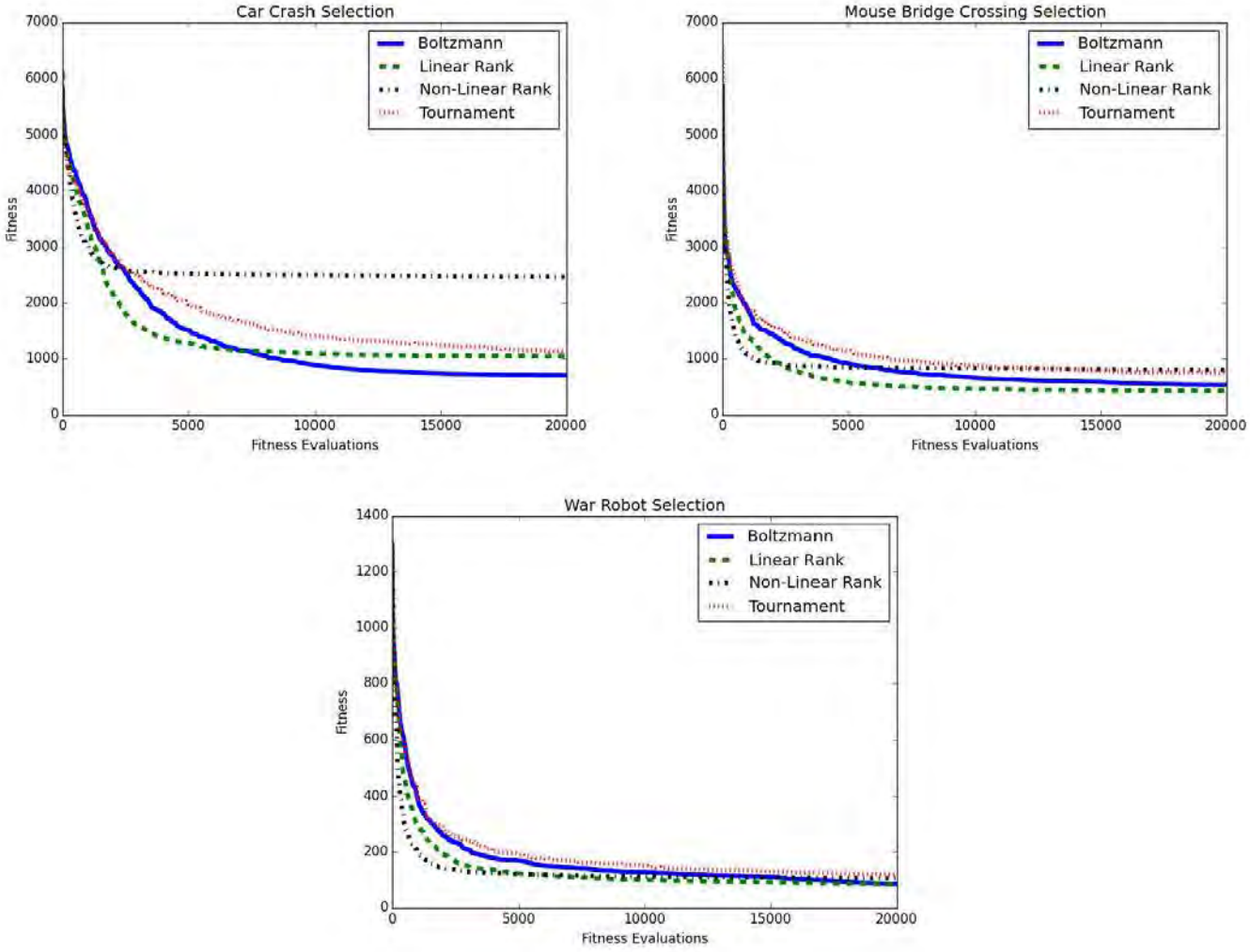

Figure 7.2: *Performance of different selection operators over 20,000 fitness evaluations in the Car Crash, Mouse Bridge Crossing, and War Robot Battle scenarios. The selection operators were run with the CNE algorithm.*

We found that although the task performance of operator probability combinations are scenario dependent, both BLX-α and Laplace crossover performed consistently better than the others, with a crossover probability of 0.6 and 0.8 performing better with these operators. For our final experiments, we have chosen to use Laplace crossover with a 0.8 probability, as it performs more consistently than the other 3 combinations.

7.2.7 Mutation Operator for CNE and ESP

We tested the three mutation operators implemented in our system using CNE in the Mouse Bridge Crossing, Car Crash, and War-Robot Battle simulation scenarios in order to determine the better performing operator (figure 7.4 and table 7.2.7). A standard deviation of 0.2 is used for Gaussian mutation, a range of $[-0.2; 0.2]$ is used for Uniform mutation, and a scale of 0.2 for Cauchy mutation. These values were obtained through experimentation.

Table 7.2: *Mean fitnesses and standard deviations achieved by each selection operator with CNE after 20,000 fitness evaluations. P-values for the one-tailed Mann-Whitney U test are given in parentheses for each operator when compared with the operator that achieved the lowest mean. A dash is given instead if the operator achieved the best mean for the specified scenario.*

Scenario	**L-Ranked**	**NL-Ranked**	**Boltzmann**	**Tournament**
MBC	**425.63 (-)**	802.86 (4.45e-10)	529.98 (2.02e-2)	738.13 (8.03e-7)
MBC σ	136.48	161.07	209.03	253.36
CC	2460.85 (8.82e-3)	1045.95 (1.33e-9)	**704.82 (-)**	1124.9 (4.28e-4)
CC σ	658.12	462.46	914.92	445.54
WRB	**86.11 (-)**	106.43 (1.31e-3)	86.61 (0.32)	120.44 (2.79e-7)
WRB σ	15.94	32.32	12.33	25.36

Table 7.3: *The mean, standard deviations, and Mann Whitney U p-values for operators that achieved the best and worst means in each scenario. A dash is given in place of a p-value if the operator was the best performing for the specified scenario.*

Scenario	**Operator**	**Mean**	σ	**P-Val**
C	UNDX 0.4	4.971	5.151	-
C	One-point 0.8	40.521	97.389	0.055
C	Arithmetic 0.6	12.780	13.852	0.006
C	BLX-α 0.8	32.253	82.412	0.08
C	Uniform 0.6	9.555	9.069	0.02
C	Laplace 0.6	5.543	8.127	0.746
MBC	Laplace 0.8	184.094	75.263	-
MBC	UNDX 0.8	490.845	171.194	4E-11
MBC	BLX-α 0.8	190.667	77.784	0.740
MBC	Uniform 0.8	209.025	72.692	0.197
MBC	Heuristic 0.4	202.239	72.456	0.345
MBC	PCX 0.2	251.795	82.851	0.001
WRB	Laplace 0.6	17.366	12.516	-
WRB	UNDX 0.8	82.866	109.375	0.002
WRB	Arithmetic 0.8	80.066	95.282	0.001
WRB	BLX-α 0.6	19.566	13.783	0.52
WRB	Uniform 0.8	28.566	22.818	0.022
WRB	Two-point 0.8	41.566	66.883	0.06

We found that Gaussian mutation was the best performer, as it performed significantly better than the other two operators in two of the simulation scenarios, and was comparable to the best performer in the third (table 7.2.7). As a consequence, we will use Gaussian mutation in our final experiment.

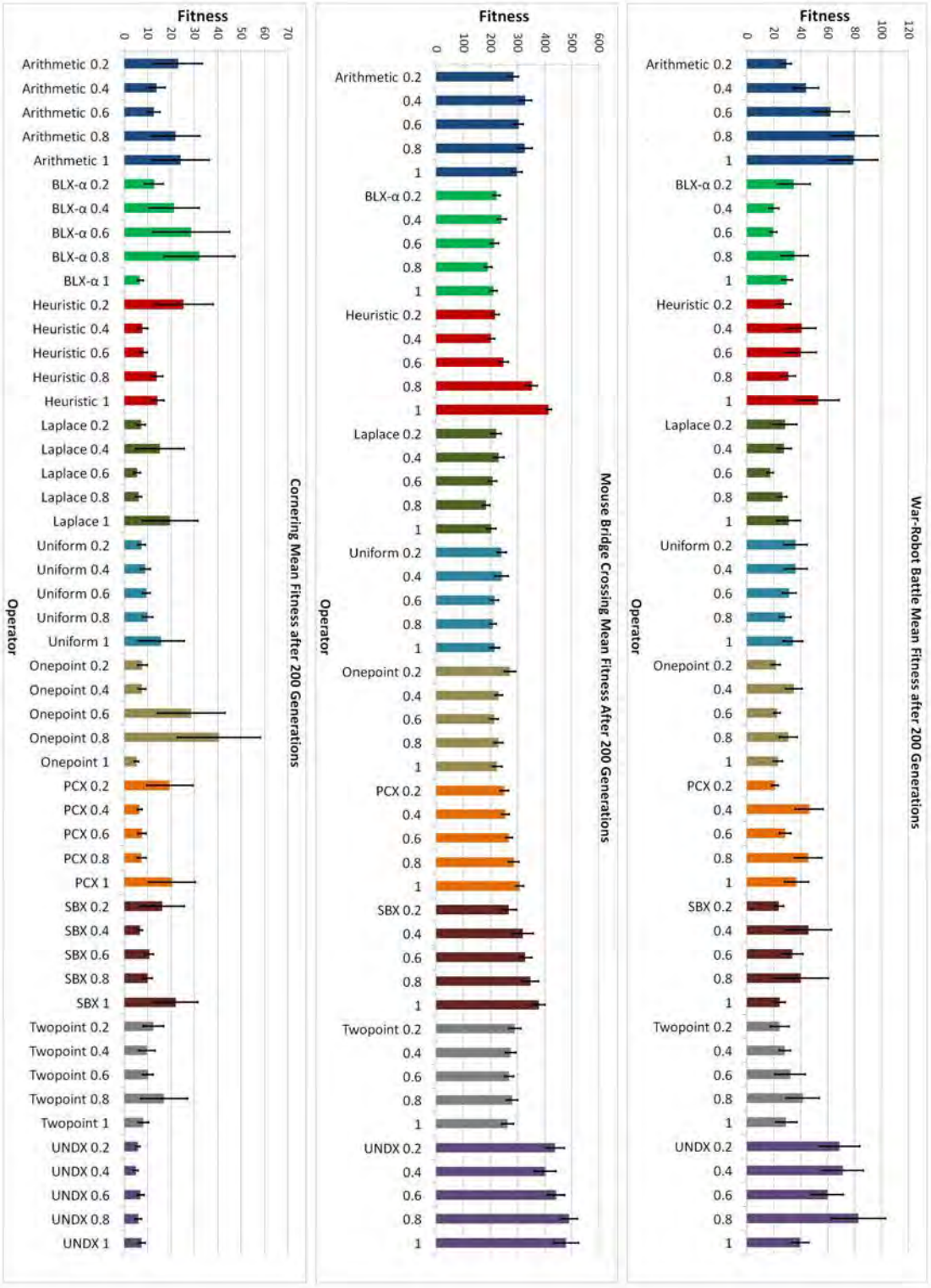

Figure 7.3: *Mean fitness of each operator probability combination across the Cornering, War Robot Battle, and Mouse Bridge Crossing scenarios. The operators were used with ESP, which was run for 200 generations. 30 evolutionary runs were performed for each combination.*

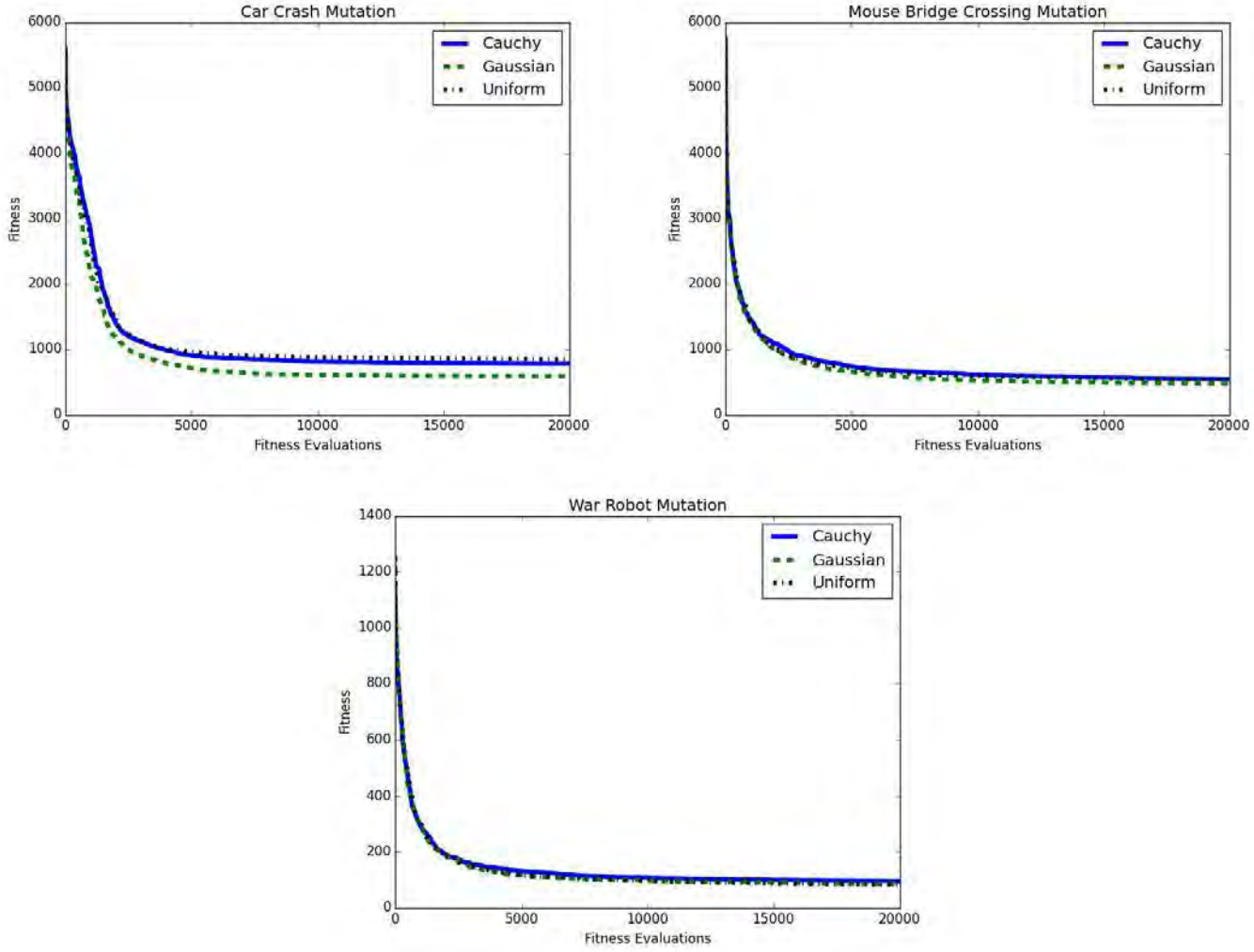

Figure 7.4: *Performance of the three mutation operators over 20,000 fitness evaluations in the Car Crash, Mouse Bridge Crossing, and War Robot Battle scenarios. As with the selection operators, the mutation operators were run with the CNE algorithm.*

7.3 Summary

This chapter describes the experiments to be run for the results chapter (chapter 8), and provides motivations and parameter tuning experiment results for the parameters and operators used in the NE algorithms.

One issue that we faced when determining the ideal parameters and operators is their sheer number. Although it would be ideal to obtain quantitative performance data for all of the parameters and operators used, it would require very extensive preliminary tests, which is infeasible for this thesis given the scope and time constraints. For this reason, we have only run preliminary experiments for some of the more important operators and parameters, with the rest of them being determined either through initial experimentation or through recommended values by the original authors.

Table 7.4: *Mean fitnesses and standard deviations achieved by each mutation operator with CNE after 20,000 fitness evaluations. the Mann Whitney U test p-values are given in parentheses, and where obtained when comparing the operator with the best performing one (which achieved the lowest mean). N/A is given instead if the operator achieved the best mean.*

Scenario	**Cauchy**	**Gaussian**	**Uniform**
MBC	544.22 (2.97e-2)	**477.87 (N/A)**	539.07 (2.03e-2)
MBC σ	136.56	208.69	128.93
CC	780.05 (8.12e-2)	**588.24 (N/A)**	847.51 (8.57e-2)
CC σ	603.16	284.69	744.95
WRB	95.87 (3.19e-3)	85.62 (0.38)	**83.82 (N/A)**
MBC σ	19.76	14.83	12.44

Chapter 8

Results and Discussion

The approach adopted by our system to control the high-level emergent behaviour of crowd simulations, the specifics in the various Neuro-Evolution (NE) algorithms, operators and parameters, and the experimental setups and designs have been detailed in the earlier chapters. In this chapter, we aim to present, analyse, and discuss our findings obtained from the experiments discussed in section 7.1, in order to answer the research questions posed in section 1.1.

As mentioned in chapter 7, the statistical test used to test for significance is the Mann Whitney U test [MW+47]. Much like in chapter 7, the labels for the scenarios in our tables are abbreviated to the first letters of their names (for example, Mouse Bridge Crossing is MBC). Full results can be found on our online repository[1].

8.1 Determining the best performing algorithm

One of the major research questions posed is *what is the best NE algorithm for controlling high-level emergent behaviours in crowd simulations?* This is important as the chosen NE algorithm can significantly impact evolution times. Moreover, it can also impact the consistency of the algorithm, resulting in a more robust evolutionary process.

In order to evaluate the best performing NE algorithm implemented in our system, we compare how well these algorithms are able to minimize the fitness function across our 11 different scenarios over 20,000 fitness evaluations.

We found CMA-ES to be the best performing algorithm, as it achieved the best mean fitness at the end of evolution in eight of the 11 scenarios, significantly outperforming all of the other algorithms in four of them, and is comparable ($p > 0.05$) to the best performer on two of them (table 8.1). The only case where it performed significantly worse than the best performing algorithm is the Mouse Escape scenario. However, this scenario proved very easy to learn for all algorithms, with the differences in mean fitness between the algorithms being negligible (table 8.1).

A possible reason for the good performance of CMA-ES is that it requires much smaller populations compared to the other algorithms, resulting in significantly less fitness evaluations per generation. This leads to CMA-ES running for many more generations, giving it more time to improve. Despite being the best performer, a problem we found

[1] https://bitbucket.org/igorawratu/thesis

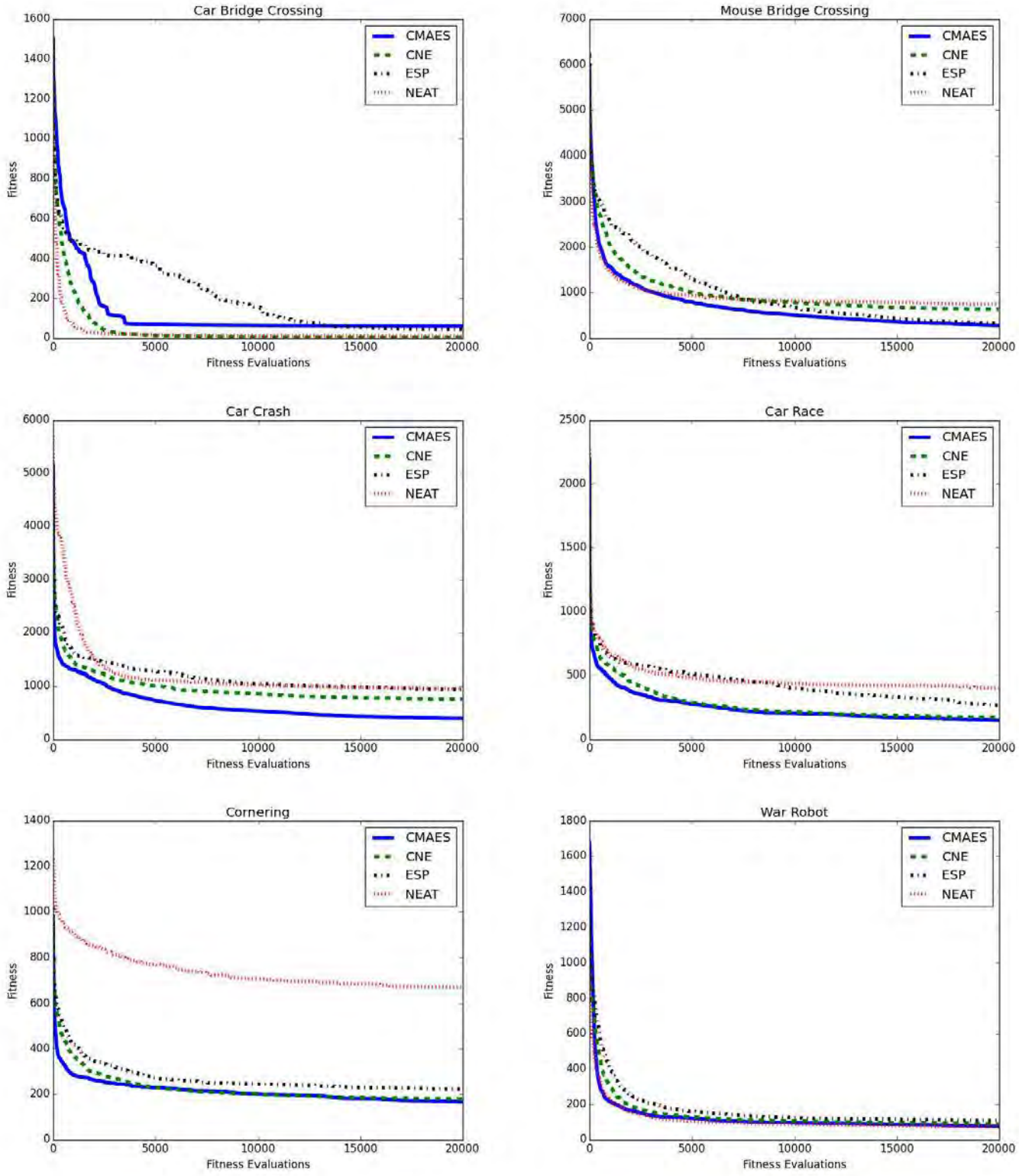

Figure 8.1: *Mean fitness of the four different NE algorithms over 20,000 fitness evaluations in the Car Bridge Crossing, Mouse Bridge Crossing, Car Crash, Car Race, Cornering, and War Robot Battle scenarios.*

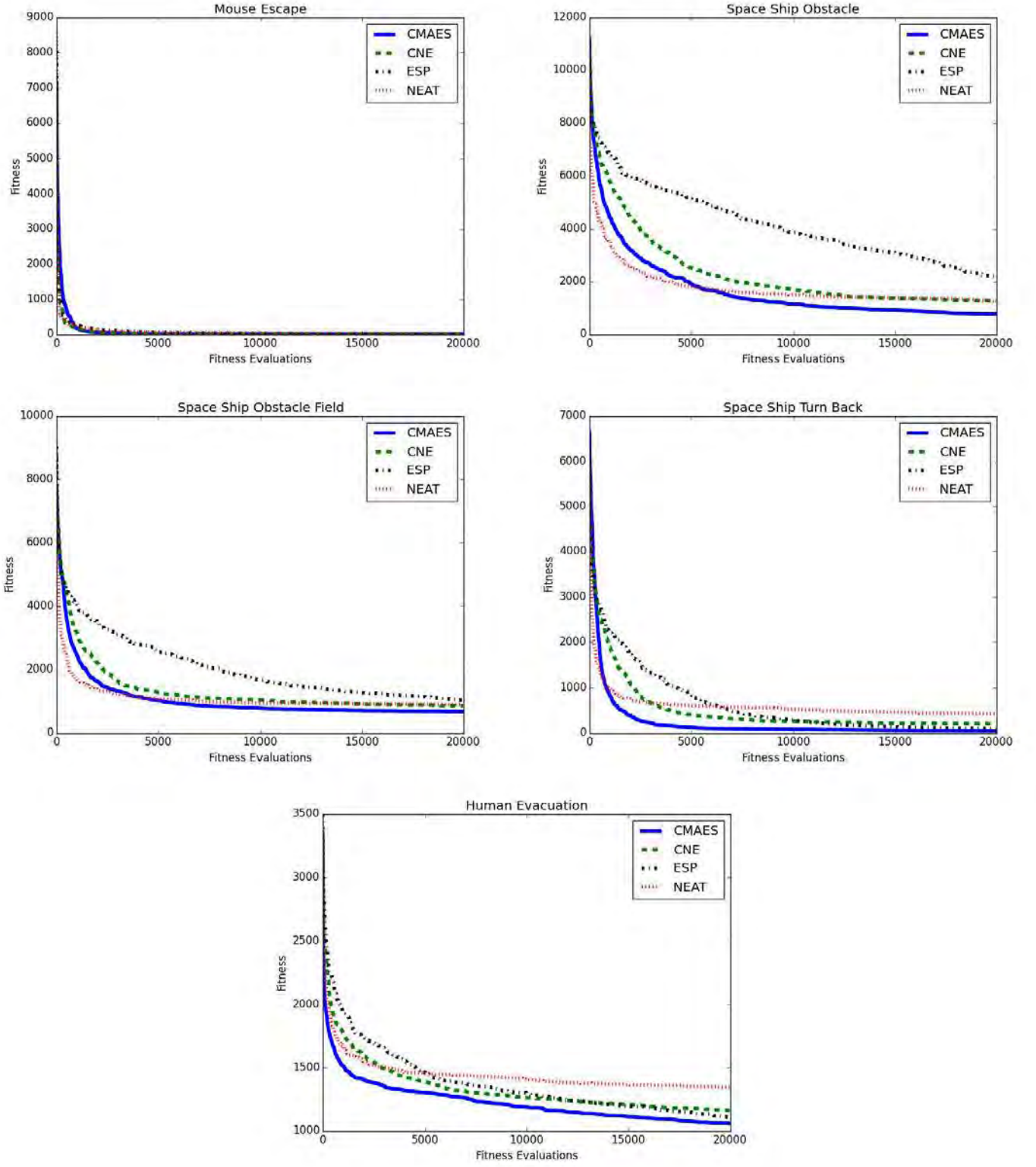

Figure 8.2: *Mean fitness of the four different NE algorithms over 20,000 fitness evaluations in the Mouse Escape, Spaceship Obstacle, Spaceship Obstacle Field, Spaceship Turn Back, and Human Evacuation scenarios.*

Table 8.1: *Mean fitness and standard deviation achieved by the NE algorithms across the various scenarios after 20,000 fitness evaluations. Fitnesses below the fitness threshold for the respective algorithm are shown in bold. Thresholds for each scenario can be found in section 6.2. P-values for the one tailed Mann Whitney U test are also given in parentheses. These p-values signify the difference between the fitnesses achieved by the current algorithm, and those achieved by the best mean fitness algorithm. A dash is given instead of a p-value in cases where the algorithm has the best mean fitness.*

Sim	**NEAT**	**CMA-ES**	**CNE**	**ESP**
CBC	**7.87(9.5e-4)**	**60.14(0.21)**	**3.35(-)**	**43.14(2.2e-3)**
CBC σ	5.73	232.75	3.43	141.19
CC	946.78(1.1e-10)	389.03(-)	752.48(3.8e-9)	935.16(1.6e-11)
CC σ	577.85	136.11	198.92	160.02
CR	401.85(1.1e-10)	**147.59(-)**	**171.34(0.47)**	265.68(6.6e-5)
CR σ	93.54	65.1	105.36	116.46
ME	**17.82(2.4e-7)**	**5.64(3.1e-6)**	**1.70(-)**	**7.75(4e-5)**
ME σ	14.2	4.9	4.22	6.87
WRB	**72.09(-)**	**74.39(0.39)**	**90.2(1.6e-4)**	**108.72(7.7e-10)**
WRB σ	15.9	10.51	18.04	15.09
STB	**428.63(6e-11)**	**47.8(-)**	**203.21(1.8e-8)**	**101.93(2.5e-8)**
STB σ	206.03	42.39	181.41	103.73
SO	1262.45(3.7e-10)	**773.99(-)**	1273.87(6.2e-5)	2190.49(5.4e-10)
SO σ	226.36	164.41	579.81	862.54
SOF	**891.21(4.5e-8)**	**675.13(-)**	**847.42(2.8e-4)**	1031.02(4.1e-7)
SOF σ	136.26	99.33	212.8	289.66
C	668.98(1.5e-11)	**165.46(-)**	**179.24(0.06)**	**221.36(1.7e-10)**
C σ	129.3	25.12	37.62	14.64
MBC	**736.39(1.5e-11)**	**271.79(-)**	**635.88(4.2e-9)**	**326.39(0.08)**
MBC σ	104.55	113.48	227.76	155.54
HE	1342.39(3.7e-10)	1057.09(-)	1162.68(3.2e-3)	1105.82(0.11)
HE σ	104.19	139.26	114.99	146.68

with CMA-ES is that its evolution times can be much more expensive compared to the other algorithms on problems with higher dimensions (table 8.1). The reason for this is the because CMA-ES makes use of numerous matrix operations (for example, inversion, multiplication, and Eigen decomposition), many of which have a computational complexity of or close to $O(n^3)$, where n is the problem dimensionality. In contrast to CMA-ES, the other implemented NE algorithms have a complexity $O(n)$, assuming that the chromosome population size stays constant.

We confirmed that these matrix operations bottlenecked the system on higher dimensional problems by running CMA-ES in the cornering scenario with various team compositions, and recording the evolution times (table 8.1). Although fitness evaluations are normally the bottleneck of the evolutionary process, it is not the case here as the

Table 8.2: *Mean evolution times of the NE algorithms over 30 runs across the various scenarios when evolved for 20,000 fitness evaluations. Standard deviations are given in parentheses.*

Scenario	**NEAT**	**CMA-ES**	**CNE**	**ESP**
CBC	280.09s (7.15s)	305.36s (3.43s)	231.31s(7.73s)	241.42s(5.17s)
CC	514.13s(11.88s)	602.51s(14.59s)	538.93s(16.45s)	531.10s(8.03s)
CR	385.83s(16.93s)	1402.17s(28.56s)	372.24s(12.24s)	356.48s(3.99s)
ME	1356.66s(508.15s)	2590.88s(68.63s)	955.60s(50.32s)	961.43s(43.86s)
WRB	2671.06s(56.19s)	5671.22s(116.11s)	2612.07s(56.78s)	2659.75s(28.11s)
STB	1086.44s(34.07s)	2201.49s(109.6s)	1013.47s(33.02s)	1017.7s(60.5s)
SO	1062.52s(25.07s)	2185.84s(26.42s)	1021.37s(34.17s)	1000.08s(18.54s)
SOF	1261.95s(74.94s)	2383.53s(39.15s)	1256.22s(75.98s)	1170.16s(20.46s)
C	221.25s(6.12s)	258.21s(4.48s)	212.04s(0.34s)	206.65s(2.21s)
MBC	913.95s(45.48s)	948.45s(20.27s)	856.78s(41.15s)	847.01s(23.68s)
HE	1023.12s(56.07s)	1074.55s(42.96s)	986.06s(21.83s)	946.7s(11.28s)

Table 8.3: *Execution times of CMA-ES for 20,000 fitness evaluations in the cornering scenario using various team compositions. This table shows that as the problem dimensionality increases, the evolution times of CMA-ES increase dramatically, in contrast to the other algorithms where the training times of the other algorithms will be much lower.*

	Homogeneous	**Semi-Heterogeneous**	**Heterogeneous**
Execution time	283.86s	1398.85s	3689.02s

evolution time increased roughly 15 times from a fully homogeneous teams composition to a fully heterogeneous when using the same number of fitness evaluations.

Despite its computational expense, CMA-ES does not necessarily need to evolve for as many fitness evaluations as the other algorithms. This is because it improves much faster than the other algorithms (figures 8.1 and 8.2). Table 8.4 shows how many fitness evaluations CMA-ES required to surpass the best mean fitness of the other algorithms on the scenarios where CMA-ES performed the best. Less than half the total number of fitness evaluations was required to surpass all other algorithms on four of the eight scenarios, showing that, in specific cases, one can set CMA-ES to evolve for a far shorter period of time and still achieve better results than the other algorithms.

On top of its excellent performance, CMA-ES also has the benefit of having most of its parameters be self-tuning. This is a major advantage as parameter tuning requires expert knowledge and trial and error, and is often scenario dependent.

The second best performer is CNE, which achieved the best mean fitness on two of the 11 scenarios, significantly outperforming the other algorithms on one of them, and is comparable to the best performer on two other scenarios (table 8.1). This is surprising as it is the simplest of the implemented algorithms, and has performed worse than the other algorithms in previous works [SM02, GM99]. Due to its simplicity, an additional advantage it has is that its computational cost scales better to higher dimensions

Table 8.4: *This table shows the number of fitness evaluations required by CMA-ES to surpass the best mean fitness achieved by the other three algorithms for the scenarios where CMA-ES performed best.*

Scenario	**Number of Fitness Evaluations in CMA-ES**
CC	4770
CR	14460
STB	5740
SO	8650
SOF	7370
C	15970
MBC	16600
HE	15420

than CMA-ES (table 8.1). This makes CNE a viable alternative to CMA-ES on higher dimensional problems, such as more heterogeneous team compositions.

NEAT performed poorly relative to both CMA-ES and CNE. However, it did still achieve the fitness threshold on six of the 11 scenarios (CNE and CMA-ES achieved eight and nine respectively), showing that it is still a viable algorithm for certain scenarios. A drawback to NEAT, which may have contributed to its poor performance, is its number of parameters, which ranges from 17 to 23 depending on the specific implementation used. This was an issue for us when conducting our experiments, as our time constraints did not allow us to tune all of these parameters. Thus, we had to settle for many of the author recommended values, which may not necessarily work well in this problem area. Furthermore, as many of these parameters are scenario dependent, users are often required to perform manual parameter tuning for each scenario.

An anomaly found when testing NEAT is that it performed very poorly in the cornering scenario, prematurely converging onto very sub-optimal solutions (cornering subfigure of figure 8.1). The reasons for this are currently unclear, and further investigation is required in order to determine the underlying causes.

Despite its drawbacks, a major benefit of NEAT is that it evolves both the topology and weights of the ANNs. This alleviates the burden of pre-determining the ideal ANN structure.

We found ESP's performance to be roughly equivalent to that of NEAT. Its poor performance can be attributed to the fact that we use a sub-population size of 50, with five fitness evaluations per neuron, leading to 250 fitness evaluations every generation. Compared to other algorithms, such as CNE and NEAT, where 100 fitness evaluations are required, or CMA-ES where even less are required, ESP will end up running for far fewer generations in the same number of fitness evaluations. As improvements only occur during transitions between generations, ESP is given much less time to improve compared to its counterparts, and thus will often not have properly converged. This can be seen in figures 8.1 and 8.2, where ESP is often still improving at the end of 20,000 fitness evaluations.

Another interesting note about ESP is that, due to its cooperative co-evolution nature, the dimensionality of each of its sub-populations is low. Thus, it may perform better on problems of higher dimensions. A future direction of research would therefore be to examine ESP in the context of more heterogeneous team compositions, when given longer evolution times.

Overall, although CMA-ES was shown to achieve the best fitness on most scenarios, its computation time on problems of higher dimensions may require the use of CNE instead. Additionally, despite the relatively poor performance of both ESP and NEAT, these algorithms are not without their benefits. NEAT, or another topology and weight evolving ANN algorithm, may be ultimately desired as it automates the work required to find the ideal ANN topology, whereas ESP may be preferable on problems of higher dimensions, due to its cooperative nature. It should also be noted that modifications to both ESP and CNE can be made to also allow for the evolution of topology, and should be investigated in the future.

8.2 Determining the best team architecture

In order to investigate more heterogeneous team compositions, we compare 4 different team setups, ranging from fully homogeneous to fully heterogeneous (section 7.1.3), for each scenario to see which can best minimize the fitness function. Although we initially intended to test these team compositions using the best NE algorithm found in section 8.1, we found that the best performer, CMA-ES, was not viable for more heterogeneous team setups. This is because of its extremely high computational cost on problems with high dimensions. For this reason, we instead use the second best performer, CNE, to investigate the various team compositions. The fitnesses obtained for the various team compositions can be found in figures 8.3 and 8.4, and table 8.2. Het, Semi-Het, Semi-Hom, and Hom in the tables presented in this section stand for Heterogeneous, Semi-Heterogeneous, Semi-Homogeneous, and Homogeneous team compositions, respectively, and refer to the different team composition types tested for.

In addition to comparing the fitness values obtained by the various team compositions, we are also looking to see how much they improve the believability of the crowd behaviour. The ideal method of achieving this is to perform qualitative tests, comparing the realism of the crowds generated using NE to crowds generated using other techniques. However, this is infeasible for us due to time and scope constraints, as performing these tests would require the implementation of a variety of techniques in our system (in order to eliminate additional factors such as animation, model, and texture quality), as well as a sizeable amount of time for conducting the tests.

Instead, we will perform an informal survey with 10 participants, where they are required to watch videos of the evolved crowd behaviours for the various scenarios and team compositions posted on our youtube channel[2], and rate between one and five on how efficient and how heterogeneous they behave, with 5 being very efficient and very heterogeneous. These ratings can be found in tables 8.2 and 8.2.

Heterogeneous team compositions were found to generally behave very diversely. However, this heterogeneous behaviour is often coupled with inefficiency, where agents often

[2] `https://www.youtube.com/playlist?list=PL1_l46LP3BfIosjqrq0SqdVSFTkEMsnIv`

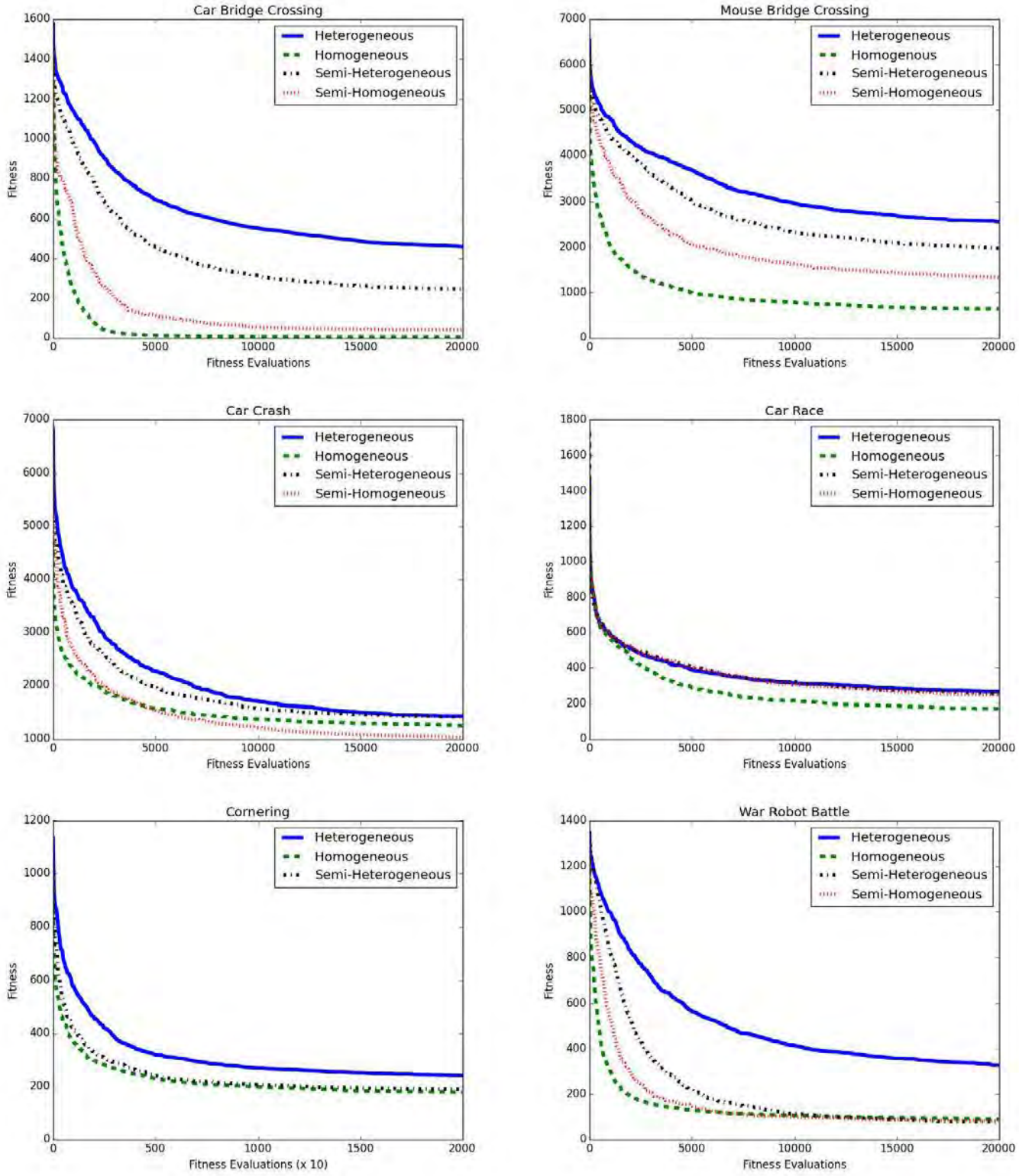

Figure 8.3: *Mean fitnesses of the 4 different team compositions over 20,000 fitness evaluations in the Car Bridge Crossing, Mouse Bridge Crossing, Car Crash, Car Race, Cornering, and War Robot Battle scenarios. The algorithm used with these compositions is CNE.*

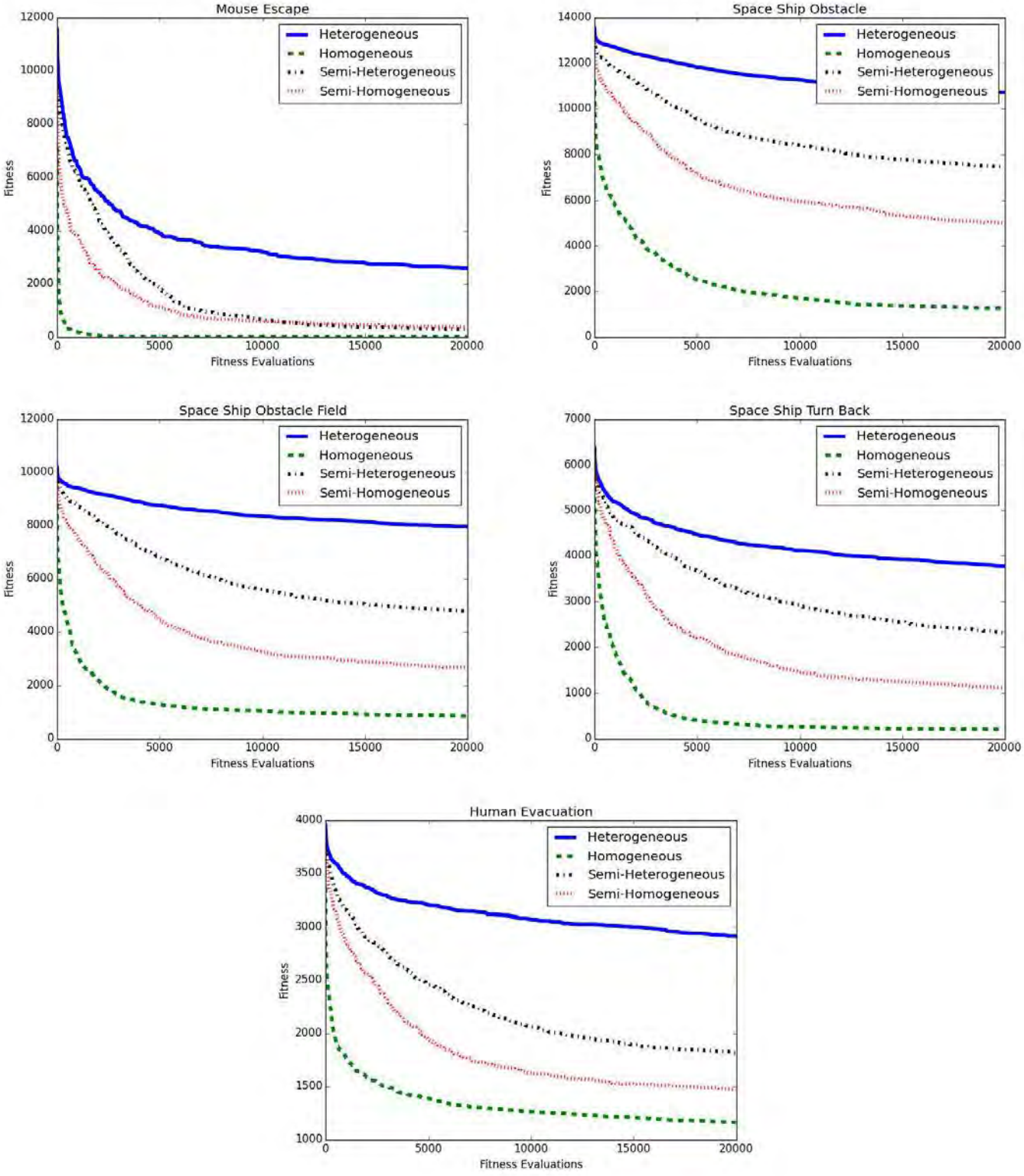

Figure 8.4: *Mean fitnesses of the 4 different team compositions over 20,000 fitness evaluations in the Mouse Escape, Spaceship Obstacle, Spaceship Obstacle Field, Spaceship Turn Back, and Evacuation scenarios. The algorithm used with these compositions is CNE.*

Table 8.5: *Mean fitness and standard deviation after 20,000 fitness evaluations achieved by the team compositions with CNE across the various scenarios. Bold fitness values achieved better values than the fitness threshold (section 6.2). P-values for the one tailed Mann Whitney U test are given in parentheses, these p-values signify the difference between the team composition's population, and the population of the team composition that achieved the best mean. A dash is given instead of a p-value in cases where the team composition has the best mean fitness.*

Sim	**Het**	**Semi-Het**	**Semi-Hom**	**Hom**
CBC	458.57(1.5e-11)	246.95(3e-8)	**41.63(0.46)**	**3.35(-)**
CBC σ	133.02	189.07	129.54	3.43
CC	1431.29(5.3e-8)	1412.7(8.5e-10)	752.48(-)	1252.37 (8.8e-3)
CC σ	501.61	318.81	308.64	373.36
CR	265.51(1.6e-4)	253.19(6.9e-4)	**245.58(2.1e-3)**	**171.34(-)**
CR σ	72.84	80.31	96.63	105.36
ME	2595.92(1.1e-11)	283.33(1.6e-10)	370.68(2.6e-8)	**1.7(-)**
ME σ	460.3	438.84	496.9	4.22
WRB	327.17(1.5e-11)	**77.13(-)**	**82.47(0.21)**	**90.2(2.1e-3)**
WRB σ	127.57	21.11	22.44	18.04
STB	3775.95(1.51e-11)	2313.23(1.5e-11)	1121.68(1.7e-10)	**203.21(-)**
STB σ	294.26	387.66	501.24	181.41
SO	10709.25(1.5e-11)	7475.86(1.5e-11)	4987.47(1.5e-11)	1273.87(-)
SO σ	420.54	789.85	1415.74	579.81
SOF	7954.81(1.5e-11)	4767.2(1.5e-11)	2647.34(2.3e-11)	**847.42(-)**
SOF σ	370.55	738.12	995.15	212.8
C	**241.52(1.4e-6)**	**189.36(0.13)**	N/A	**179.24(-)**
C σ	46.2	31.33	N/A	37.62
MBC	2556.75(1.5e-11)	1971.75(1.5e-11)	1336.87(2.7e-9)	**635.88(-)**
MBC σ	308.75	335.37	471.96	227.76
HE	2912.85(1.5e-11)	1815.94(2e-11)	1471.37(1.7e-10)	1162.68(-)
HE σ	88.8	199.69	131.29	115

moved in the incorrect direction or just did not move at all. This caused the crowd behaviour to be very unrealistic. Furthermore, crowds with heterogeneous team compositions often achieved very poor fitness ratings, with the NE algorithm evolving behaviours that are very different from the desired ones. This is largely expected as the problem dimensionality increases dramatically as the team composition becomes more heterogeneous.

There are two scenarios that are exceptions to this general trend for heterogeneous team setups: Cornering; and Car Race. In these scenarios, the heterogeneous team compositions achieved moderate efficiency ratings. The reasons for this is two-fold: firstly, there are a small number of agents within the scenarios (four and 10 for the cornering and car race scenarios respectively), resulting in a relatively low problem dimensionality; secondly, the environments are narrow and restrictive, resulting in unorthodox behaviours

Table 8.6: *Mean heterogeneity ratings of the various team compositions across our 11 scenarios, with 5 being very heterogeneous, and 1 being very homogeneous. Standard deviations are given in parentheses.*

Scenario	**Het**	**Semi-Het**	**Semi-Hom**	**Hom**
CBC	4.3 (0.67)	3.6 (0.7)	2.7 (0.82)	1.3 (0.48)
CC	3 (0.82)	2.8 (0.63)	2.3 (0.82)	2.9 (1.1)
CR	3.6 (0.84)	2.9 (0.88)	2.7 (0.82)	1.9 (0.99)
ME	4.2 (1.14)	3.6 (0.97)	3 (0.94)	2.2 (0.63)
WRB	3.6 (1.51)	3.1 (0.99)	2.9 (0.88)	2.4 (0.84)
STB	4.5 (0.71)	3.8 (0.42)	2.7 (0.48)	1.9 (0.57)
SO	4.5 (0.85)	4.2 (0.79)	3.5 (0.97)	1.6 (0.7)
SOF	4.3 (1.06)	3.7 (0.95)	3.1 (0.99)	1.5 (0.71)
C	3.4 (1.07)	3 (0.67)	N/A	2 (0.82)
MBC	4.6 (0.52)	4 (1.05)	3.1 (0.87)	2.1 (0.74)
HE	4.5 (0.71)	3.9 (0.74)	3.3 (0.95)	1.8 (0.92)

being much more difficult to evolve than orthodox ones (for example, moving in the incorrect direction requires the agent to slow down enough so that it is able to turn around).

The semi-heterogeneous team compositions performed very similarly to the heterogeneous ones, except with better efficiency, slightly worse heterogeneity, and slightly better fitness. The primary reason that these team compositions achieved a worse heterogeneity rating than the heterogeneous one is that it is possible to see groups of agents synchronize their movements in scenarios with very sparse environments (such as spaceship turn back). Additionally, despite it having slightly better fitness and efficiency ratings, the crowd behaviours were still too poor to be of any use. However, we did find that its fitness in the War-Robot scenario greatly improved compared to the heterogeneous team composition, having actually achieved the best mean fitness amongst the various team compositions.

The semi-homogeneous team compositions performed respectably, achieving the fitness thresholds on three of the simpler scenarios, and receiving moderate ratings for both heterogeneity and efficiency. Additionally, despite it not achieving the fitness threshold for the car crash scenario, it was the best performer amongst the team compositions tested. We found that this team composition had a similar group synchronization issue as the semi-heterogeneous one, except more pronounced due to the fewer and larger groups.

We found that the homogeneous team compositions tend have high efficiency ratings, and low to moderate heterogeneity scores, and low (lower is better) fitness scores. The high efficiency ratings are primarily because the agents actually evolved the desired behaviours, which required efficient behaviours because of the time limit given to each scenario. Additionally, despite receiving low heterogeneity ratings, it is not always desirable, or possible, for agents to behave heterogeneously in order for the crowd to appear believable. Examples of this include the bridge crossing scenarios, where the general optimal behaviour would be to have agents move forward and converge as they approach the bridge, leading to rather homogeneous behaviours, and the car race and cornering

Table 8.7: *Efficiency rating of the agents' behaviours for the various team compositions across our 11 scenarios, with 5 being very efficient, and 1 being very inefficient. Standard deviations are provided in parentheses.*

Scenario	**Het**	**Semi-Het**	**Semi-Hom**	**Hom**
CBC	1.9 (1.29)	2.7 (1.06)	4.3 (0.67)	4.8 (0.42)
CC	2 (0.94)	3.4 (0.97)	4.1 (0.99)	3.3 (0.95)
CR	2.9 (0.74)	3.5 (0.53)	3.2 (0.79)	4.4 (0.7)
ME	2.1 (0.74)	3.5 (0.71)	3.8 (0.79)	3.7 (1.16)
WRB	2.4 (1.35)	2.8 (1.23)	3.6 (0.7)	3.9 (0.88)
STB	1.2 (0.42)	2.1 (0.57)	3.1 (0.88)	4.3 (0.67)
SO	1.1 (0.32)	2.2 (0.63)	2.5 (0.71)	3.9 (1.1)
SOF	1 (0)	2.3 (0.67)	2.7 (0.95)	4.7 (0.48)
C	3.4 (0.52)	3.5 (0.85)	N/A	4.5 (0.71)
MBC	1.7 (1.25)	2.2 (0.79)	3.6 (0.52)	4.5 (0.53)
HE	1.5 (0.71)	2.5 (0.85)	2.5 (0.85)	3.8 (1.14)

scenarios, where there is little space for the agents to deviate substantially in their behaviours. We found that for many scenarios, the agent behaviours obtained by providing a homogeneous teams controller with local agent inputs were sufficiently heterogeneous.

The scenarios where the homogeneous teams composition had an impact on the believability were primarily the Spaceship scenarios, and the War Robot Battle scenario. The agents in the Spaceship scenarios tend to keep close together and move in the same direction, leading to the crowd behaving more akin to a school of fish or a bird flock, as opposed to a group of small spaceships, where the behaviours are much more chaotic (as current films would have us believe). Additionally, due to the homogeneity of the space ships' behaviour, they do not evolve scattering and merging behaviour, which can be observed in real-life groups such as bird flocks when encountering obstacles. In the War Robot Battle scenario, we found that the agents tend to all move in the same direction. This is unrealistic as the agents' direction of movement should ideally be uniform when there is no objective to move towards. The overly homogeneous behaviours (as can be seen in all the homogeneous space ship simulations, as well as the war robot one) in these scenarios were mainly caused by the environment being sparse, which leads to many of the agents having similar local inputs, resulting in the agent behaviours synchronizing.

Overall, we found that the more heterogeneous the team setups, the poorer the efficiency and fitness ratings. However, these results may possibly be biased, as the videos that the users had to watch for the survey were labelled with their categories. Although we did not state explicitly what the categories meant, it is highly possible that the users picked up on the correlation between the ratings and the categories.

A contributing factor to the poor efficiency and fitness achieved for more heterogeneous team compositions is that we do not scale the population size of CNE for the more heterogeneous team compositions. The reasons for this is because we wanted to keep the number of generations and fitness evaluations consistent across different team compositions, and that we wanted to keep the training times feasibly low. This is a problem as a population size that may sufficiently cover the search space in a homogeneous teams setup

Table 8.8: *Success rates and mean execution times of the successful evolution runs for our 11 scenarios, with 30 evolution runs being performed for each scenario. The algorithm used was CNE, and the team compositions used were the fully homogeneous compositions. Also shown are the training times, and the times expert users took to tune parameters in Jacka's work. In these times, FAIL signifies failure to train, and N/A signifies that there is no corresponding scenario. Standard deviations are given in parentheses.*

Sim	**Success**	**Mean Success time**	**Jacka's time**	**Expert user time**
CBC	100%	25.46s(8.15s)	3487.74s	2880s
CC	3.3%	336.4s(55.45s)	FAIL	4020s
CR	70%	680.07s(513.08s)	1402.26s	1260s
ME	100%	88.62s(42.36s)	7699.41s	3420s
WRB	100%	446.94s(235.76s)	7889.93s	6660s
STB	100%	114.54s(67.18s)	346.12s	840s
SO	86.7%	1198.68s(423.35s)	FAIL	1020s
SOF	100%	625.57s(290.4s)	7381.09s	3600s
C	100%	76.05s(58.07s)	FAIL	1200s
MBC	96.7%	702.22s(447.25s)	1414.88s	1200s
HE	20%	1460.73s(187.98s)	N/A	N/A

may not for more heterogeneous ones, causing the population to converge to sub-optimal solutions.

Another contributing factor is due to us using CNE, as it may scale poorly with increasing problem dimensionality because it represents and evolves entire candidate solutions as monolithic vectors, resulting in the problem of *two steps forward, one step back* [VdBE04]. This problem is characterized by the fact that while certain components of the ANN are moving closer to the optima, others can move away so long as the candidate solution provides an improvement over previous ones.

In order to better investigate how more heterogeneous team setups impact the behaviour of crowds, cooperative co-evolution NE algorithms such as ESP [GM03b], CoSyNE [GSM06], and SANE [MM96] should be further investigated. The divide and conquer approach these algorithms adopt allows for the dimensionality of the sub-populations to remain constant when the controller setup becomes more heterogeneous, as they merely add more sub-populations for the additional neurons or synapses. However, it should be noted that the addition of sub-populations also requires more fitness evaluations per chromosome in order to reduce the amount of noise.

8.3 Determining Feasibility

Table 8.3 shows the success rates and the mean evolution times of the successful runs of the 11 scenarios, where success is determined by whether a specific evolution run reached the fitness threshold in under 20,000 fitness evaluations. Thirty evolution runs were performed for each scenario, with the best performing NE algorithm and the homogeneous team composition being used. Additionally, it also shows the training times achieved

by Jacka's method [Jac09], and the times expert users required to tune the Fuzzy Systems using Jacka's system in order to achieve the desired crowd behaviours for similar corresponding scenarios.

We found that the six easiest scenarios to evolve for were the Cornering, Car Bridge Crossing, Mouse Escape, War Robot Battle, Spaceship Turn Back, and Spaceship obstacle field, all achieving 100% success rates. These scenarios typically either had very straightforward objectives with very few obstacles (Car Bridge Crossing, Cornering, Spaceship Turn Back), or were aimed at controlling agent group sizes (War Robot Battle, Mouse Escape). We also found that although the Spaceship Obstacle Field scenario had many obstacles, agents did not need to deviate much from their original path in order to avoid them, resulting in an easier to learn behaviour as the trajectories were straightforward.

We found three scenarios of moderate difficulty: the Car Race; Spaceship Obstacle; and Mouse Bridge Crossing scenarios. These had very high success rates, but did not reach 100%. These generally had more difficult objectives (Car Race), or more challenging environments (Spaceship Obstacle, Mouse Bridge Crossing). The Car Race scenario was difficult because having the underdog overtake the other agents often results in collisions due to the narrowness of the race track. The Space Ship Obstacle scenario required the agents to navigate around the large wall before the goal, which required the crowd to learn an ideal trajectory in order to not overshoot the goal point or collide with the obstacle. Finally, the Mouse Bridge Crossing scenario was difficult as it has a larger number of agents compared to the Car Bridge Crossing scenario, moving through a bottleneck of the same size. Additionally, the halting behaviour of the mice reduces their average speed when compared to car agents.

The Car Crash and Human Evacuation scenarios were the most difficult, achieving success rates of close to 0%. The difficulty of the Car Race scenario can be attributed to the narrowness of the corridor, and the close proximity of the agents. This results in the agents having to learn near perfect movement in order to avoid collisions. The main difficulty with the Human Evacuation scenario was the *die near the entrance* objective, as the agents struggled to move close enough to each other before they escaped, resulting in the inability to exert the pushing behaviour. This lead to too many agents being left alive at the end of the scenario.

As shown, it is certainly possible to use NE to control emergent aggregate crowd behaviours. However, as the task environment and objectives become increasingly restrictive, the chances of the evolutionary process finding a solution greatly diminishes. One possible reason for this is that we use the ANNs to directly control the low-level motor behaviours of the agents. Another approach would be to instead use a hierarchical behavioural model [RMT01, PAB07], where an agent's high-level cognition and decision making is separate from its low-level motor functions. The ANN would then instead be responsible for controlling this higher level, rather than the direct motor behaviours. This may allow for more complex and difficult behaviours to be evolved, whilst keeping the dimensionality of the search space low. A possible downside to this approach, however, is that the additional layers of indirection may cause the crowd behaviour to be more difficult to control.

In order to determine whether or not using NE to control emergent crowds is feasible, we compare our mean evolution times with both the mean training times achieved in Jacka's method [Jac09], and with the parameter modification times that expert users re-

quired whilst using Jacka's system (table 8.3). These times are not directly comparable, as our scenarios differ both in implementation and hardware used. Additionally, Jacka uses Fuzzy Systems as agent controllers, and Particle Swarm Optimization for training, whereas we use ANNs as agent controllers, and Evolutionary Algorithms (EAs). A last difference is that he checked for the complete satisfaction of objectives, whereas we only check if the fitness reaches a threshold. The reason for this is that his work modifies the membership functions of an existing Fuzzy Controller structure, which already largely defines the rule-set for agents, whereas ANNs do not have any of this pre-defined knowledge. This meant that we had to add in additional objectives to help guide the ANNs to evolve the desirable behaviours, such as *minimize angular velocities* and *die near the exit*, which do not have to be perfectly satisfied in order for the crowd to behave as desired.

Despite the differences, the training and expert user parameter modification times provide us with a rough guideline as to what the acceptable evolution times are for our system. We find that our evolution times are roughly 2-100 times faster than Jacka's for every scenario, and were also able to successfully evolve certain scenarios for which his system failed (Cornering, Spaceship Obstacle). Additionally, we also found that our evolution times were much faster than the times expert users required to modify the Fuzzy Systems when using Jacka's system to achieve similar results. The only scenario in which the user parameter modification time was faster was the Spaceship Obstacle scenario. This shows that it is feasible to use NE to control emergent crowd behaviours, as even for longer evolution times, the user is still freed up so they can perform other tasks.

Although shown to be feasible, it should be noted that these evolution times were acquired using algorithms, parameters, ANN structures, and objective weightings that were pre-determined. This is a serious drawback to the system as many of these are nonsensical to a user that is not well versed in EAs. Moreover, even if a user were to have expert knowledge of EAs, the amount of time it takes to find these ideal parameters is too long to be feasible. Although our work partially deals with this by determining algorithms and parameters that tend to work well across the various scenarios, many of the settings remain problem specific. However, it should be noted that both CMA-ES and NEAT are capable of partially alleviating this issue, as NEAT automates the creation of an ANN's topology, and CMA-ES has a large array of self-adapting parameters. However, further work, such as investigating the use of Pareto-based multi-objective optimization algorithms [Coe99], is required to remove the tuning required for the other parameters.

8.4 Evaluating scalability

Table 8.4 shows the evolution times of the various scenarios when one to four slave processes are used on our machine[3]. CNE was used for obtaining these times due to its low overhead.

We found that the evolution times roughly scaled linearly when adding processes (table 8.4). The exception here is the Cornering scenario, where the evolution time using four processes was roughly equivalent to the evolution time using three. The reason for this is because the Cornering scenario is very computationally cheap. Thus, the bottleneck

[3]Intel i5-3570, 3.4GHz

Table 8.9: *Evolution times of 20,000 fitness evaluations for the various scenarios when one to four slave processes are used. The algorithm used for this is CNE due to its low overhead.*

Scenario	**1 slave**	**2 slaves**	**3 slaves**	**4 slaves**
CBC	959.86s	445.41s (2.16x)	295.05s (3.25x)	263.68s (3.64x)
CC	1903.55s	963.13s (1.98x)	639.29s (2.98x)	496.03s (3.74x)
CR	1303.04s	664.98s (1.96x)	471.48s (2.76x)	360.93s (3.6x)
ME	3602.48s	1872.47s (1.92x)	1188.13s (3.03x)	1100.38s (3.27x)
WRB	10131.63s	5180.78s (1.96x)	3568.2s (2.84x)	2875.77s (3.52x)
STB	3667.34s	1988.41s (1.84x)	1274.22s (2.89x)	1126.54s (3.26x)
SO	3642.13s	1862.22s (1.96x)	1306.83s (2.79x)	1141.88s (3.19x)
SOF	4438.95s	2343.39s (1.89x)	1526.56s (2.91)	1314.25s (3.38x)
C	569s	305.67s (1.86x)	212.33s (2.68x)	210.81s (2.7x)
MBC	3363.23s	1598.52s (2.1x)	1045.19s (3.22x)	895.47s (3.76x)
HE	3224.19s	1661.83s (1.94x)	1093s (2.95x)	870.26s (3.71x)

occurs on the master process, causing the slave to often idle and wait. Additionally, there is also a slight slow down in scaling when using 4 slave processes, as the evolution time is only roughly 3.5 times that of one slave process. The reason for this is because there are only four cores, whereas there are four slave processes and one master process. This results in some processes having to share the same CPU core.

This system can be scaled to n processes, where n is the number of fitness evaluations performed every generation in the EA. Although these tests were conducted on a single machine, we plan on modifying the system in the future so that it can run on a cluster.

8.5 Summary

This chapter addressed the research questions posed throughout this thesis. We compared the four implemented NE algorithms across our 11 scenarios, and found CMA-ES to be the best performer, as it both achieved the best mean fitness after 20,000 fitness evaluations, improved the fastest, and uses a wide array of self-adapting parameters. However, despite its high performance, it becomes computationally costly as the problem dimensionality increases. The other algorithms are not without their respective benefits. CNE performed the second best, and is computationally cheap and simple to implement, NEAT automates the ANN topology, allowing for an easier end-user experience, and ESP's cooperative co-evolution nature makes it a viable avenue for further investigation in higher dimensional problems.

We also compared four different scenario specific team compositions for each of our 11 scenarios. We found that in most scenarios, homogeneous teams is sufficient for believable scenes as the agents are able to derive different behaviours solely through the use of local inputs. However, in scenarios with sparser environments, it may be better to use a more heterogeneous setup, as the movements of agents in the homogeneous setups tend to synchronize due to many of their inputs being similar. We also found that as the team

compositions became more heterogeneous, the fitness of the trained solutions worsened. This leads to the crowds exhibiting poor behaviour for heterogeneous team compositions, as they are unable to accomplish their objectives. This poor fitness is primarily because more heterogeneous team setups significantly increase the problem dimensionality, resulting in them being more difficult to evolve. Another contributing factor is that we used CNE for these experiments, which may not be the most ideal algorithm as it does not have any mechanisms to deal with increasing problem dimensionality. We also did not scale the chromosome population size, as we wanted to keep the generations and fitness evaluations constant, and the evolution time low. This resulted in the population of chromosomes being insufficient for properly covering the search spaces when the problem dimensionality increased. In order to better investigate the various team compositions, more cooperative co-evolution algorithms such as ESP, SANE, and CoSYNE should be looked into.

In order to evaluate whether it is possible to use NE to control emergent high-level crowd behaviours, and if the evolution times are feasible for use in films, we tested to see how often the NE algorithm could reach the fitness threshold for each scenario, and compared the mean training times of these successful runs to both the mean training times in Jacka's [Jac09] method, and the expert user parameter modification times when using Jacka's system. We found that NE was able to reach the fitness threshold consistently on nine out of 11 scenarios, with six of those scenarios achieving 100% success rates. The reason that NE achieved low success rates in two of the scenarios is primarily because of the highly restrictive environments and objectives. We also found the training times of these successful runs to be very promising, with the mean evolution times being faster than those in Jacka's method, and in many cases than those required by expert-users.

Another aspect of our system that we evaluated is its scalability. We found that the system scales roughly linearly on a single machine, with the maximum number of processes being the number of fitness evaluations performed every generation. Future work would be to modify the system in order to evaluate its performance on a cluster.

Chapter 9

Conclusion

In this thesis, we have provided a preliminary study into using Neuro-Evolution (NE) to control the emergent aggregate behaviour of agent-based crowd simulations. The implemented system evolves Artificial Neural-Networks (ANNs), which are used as the agent-controllers in our scenarios, in order for the aggregate emergent crowd behaviour to satisfy a set of objectives specified by the user. Compared to existing agent-based control techniques, this method is advantageous because it allows for a much wider array of agent and objective types, and it requires less user interaction, enabling easier authoring and freeing up the artists' time, allowing them to perform other tasks while evolution is occurring. Overall, we have shown that NE is a viable solution for most of the simulation scenarios provided. However, further studies should be performed in order to investigate its viability with larger sized crowds, such as the ones used in films.

9.1 Application and Contributions

The main contribution of this thesis is a novel method for automating high-level control in crowd simulations by using NE, with the main potential application area being animating crowds in films. This is important as the current film industry standard for crowd simulation, *Massive*, requires artists to spend large amounts of time and effort manually constructing and modifying the agents' brains and goals in order to achieve the desired high-level crowd behaviour.

As NE has not, to our knowledge, been used before to control high-level crowd behaviour, much of the work in this thesis is aimed at exploring and evaluating the various NE algorithms and parameters in this problem space. This is important as it gives us an idea of which algorithmic and parameter choices perform well in this context, which will allow for a faster and more accurate adaptive process. In order to achieve this, we implemented and compared four different NE algorithms, each belonging to a different class.

We found CMA-ES to significantly outperform the other algorithms on most scenarios in terms of both improvement speed, and final task performance. CMA-ES also has the benefit that most of its parameters are self-adapting, which allows for an easier authoring process as users to not have to manually tune the parameters for the relevant scenario. However, we did find that its computational cost increased greatly on higher dimensional problems, leading us to a preference for other algorithms, such as CNE, in such cases.

Another aspect investigated in this thesis is how various controller team compositions impact both the evolutionary process, and the behaviour of crowds. From these experiments, we found that homogeneous team compositions are able to obtain sufficiently diverse agent behaviours in most simulation scenarios. However, when a scenario's environment is sparse, a homogeneous crowd will tend to have agents with synchronous behaviours, leading to an unrealistic scene. In these cases, it is better to use a more heterogeneous approach so that the agents vary their behaviours. However, it was shown that as crowds become more heterogeneous, they satisfied the user-specified objectives less effectively, leading to agents adopting very poor and often unorthodox behaviours. We did however discover that, in the Car Crash scenario, a semi-homogeneous approach where agents were divided into groups achieved the specified goals better than a fully homogeneous one, showing that a more heterogeneous approach is, in rare cases, more effective at evolving the desired behaviours.

9.2 Limitations and Future Work

There are currently several major limitations and exclusions to the system presented in this thesis. These, together with possible directions of future research, are further elaborated in this section.

9.2.1 Lack of an Authoring System

As this thesis mainly deals with exploring the various NE mechanisms to control emergent aggregate crowd behaviours, no authoring system was implemented. The agents, scenarios, and ANN output interpretations are hard-coded, making the system unsuitable for an end-user. A future direction would be to create an authoring system that allows for easy definition and creation of agents and simulations, and for easy specification of how the ANN outputs relate to the agents' behaviours.

9.2.2 Automating NE parameters

Due to the scenario-specific nature of many of the NE parameters and algorithms, obtaining the ideal evolution times and performance requires a lot of trial and error. This is undesirable for an end-user as it requires both expert knowledge and time. Although our work deals with this in part, by attempting to find some of the best overall settings, many of these settings (such as weightings for objectives) are very specific to the context and cannot be generalized. Thus, it is necessary to investigate methods for automating the tuning of these parameters.

One such algorithm is CMA-ES, which self-adapts its parameters according to the number of dimensions for the specific problem. However, CMA-ES has the drawback that it is computationally expensive on problems of higher dimensions. Thus, it is not a feasible solution in these cases.

Another NE algorithm that adapts parameters is Neuro-Evolution of Augmenting Topologies (NEAT), which allows for the automatic creation and tuning of ANN topologies, as it evolves both the topologies and weights of the ANNs. However, due to its poor performance compared to the other algorithms, it is necessary to investigate methods for

improving its performance when used to control crowd simulations. NEAT also has the disadvantage that it contains a very large number of parameters, ranging between 17 and 23 depending on the specific implementation used.

A class of methods that may help with removing the need for objective weightings is Pareto-Based Multi-Objective Optimization algorithms [Coe99], which attempt to evolve a population of candidate solutions lying on the Pareto Front [BT80] of the various objectives. Additionally, solutions on the Pareto front may be used in a more heterogeneous setting in order to generate more diverse behaviours amongst the agents.

Finally, finding or developing an algorithm that generalizes well to problems of higher dimensions will bypass the need for switching out algorithms based on the problem dimensionality (for example, CMA-ES performs best on low to medium dimension simulations, but becomes too computationally expensive on higher ones). For this, cooperative co-evolution NE algorithms are promising due to their divide and conquer approach, which may be more suitable for simulations of higher dimensionality as the sub-population dimensionality remains small.

An alternative approach to finding a general NE algorithm that self-adapts all its parameters is to use *hyper-heuristics* algorithms [BGH+13]. In contrast to meta-heuristics algorithms, such as EAs, which search within the search-space of the problem space, hyper-heuristics algorithms aim to instead search within a search-space of heuristics aiming to find the correct heuristic, or sequence of heuristics, to solve a problem. Thus, hyper-heuristics can be used to solve a class of problems, rather than a specific one [BGH+13]. This can potentially be applied to our system by treating parameter setups as different heuristics, with the aim of the hyper-heuristics algorithm being to find the ideal parameter setup.

Another parameter tuning approach that should be investigated in the future is the F-Race and iterated F-Race algorithms [BYBS10], which aim to select the best configuration out of a set when under stochastic evaluations.

9.2.3 Training for Complex Objectives and Environments

A problem we found with the method described in this thesis is that it struggles to fully achieve the given objectives when either the objectives or the environment are too restrictive. This may be in part due to the use of ANNs to directly control agents' low-level motor behaviours, as many of the behaviours required to accomplish these objectives may be very difficult to learn.

In order to combat this, the use of ANNs in hierarchical behavioural models [RMT01, PAB07] should be investigated, where the ANNs instead control the higher level cognitive and goal functionalities of the agents. This would make it possible to define lower level behaviours algorithmically, allowing for more complex behaviours to be learned. Further in-depth investigation into the user-specified objectives is also helpful as it allows for more interesting crowd behaviours and faster evolution.

9.2.4 Accelerating Training

Although the evolution times in our method are acceptable, they can still be improved. This is illustrated in some of our more complex simulations, where users are required to

wait roughly two hours before a solution is found. Thus, a future direction of research would be to investigate methods of accelerating the evolutionary process.

Although we used a distributed framework to implement our system, we have only tested its performance using multiple processes on a single machine. Thus, a future step is to adapt and investigate the system's performance on clusters. Another approach that can be used to accelerate the system is investigating the implementation of a GPU accelerated version of the scenarios, and CMA-ES.

The benefits for accelerating the system are that users will have to wait less, simulations can have larger populations of agents and run for longer periods of time, and larger populations of chromosomes can be evolved in the NE algorithm.

Bibliography

[ADA94] Ram Bhusan Agrawal, Kalyanmoy Deb, and Ram Bhushan Agrawal. Simulated binary crossover for continuous search space. 1994.

[AK88] Emile Aarts and Jan Korst. Simulated annealing and boltzmann machines: a stochastic approach to combinatorial optimization and neural computing. 1988.

[AMC03] Matt Anderson, Eric McDaniel, and Stephen Chenney. Constrained animation of flocks. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 286–297. Eurographics Association, 2003.

[And89] Charles W Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37, 1989.

[ATL+08] Alexandros Agapitos, Julian Togelius, Simon M Lucas, Jürgen Schmidhuber, and Andreas Konstantinidis. Generating diverse opponents with multiobjective evolution. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 135–142. IEEE, 2008.

[Bab98] Robert Babuska. *Fuzzy modeling for control*. Kluwer Academic Publishers, 1998.

[Bac94] Thomas Back. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 57–62. IEEE, 1994.

[Bak85] James Edward Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms and their applications*, pages 101–111. Hillsdale, New Jersey, 1985.

[Bal98] Tucker Balch. Behavioral diversity in learning robot teams. 1998.

[BBM05] Adriana Braun, Bardo EJ Bodmann, and Soraia R Musse. Simulating virtual crowds in emergency situations. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 244–252. ACM, 2005.

[BBZ05] Gunnar Buason, Nicklas Bergfeldt, and Tom Ziemke. Brains, bodies, and beyond: Competitive co-evolution of robot controllers, morphologies and

environments. *Genetic Programming and Evolvable Machines*, 6(1):25–51, 2005.

[BG95] Bruce M Blumberg and Tinsley A Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In *Proceedings of ACM SIGGRAPH*, pages 47–54. ACM, 1995.

[BGH+13] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[BH97] David C Brogan and Jessica K Hodgins. Group behaviors for systems with significant dynamics. *Autonomous Robots*, 4(1):137–153, 1997.

[BHD94] R Beckers, OE Holland, and Jean-Louis Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In *Artificial life IV*, volume 181, page 189, 1994.

[BK11] Zafer Bingül and Oğuzhan Karahan. A fuzzy logic controller tuned with pso for 2 dof robot trajectory control. *Expert Systems with Applications*, 38(1):1017–1031, 2011.

[BL07] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34:1–41, 2007.

[BLA02] O Burchan Bayazit, Jyh-Ming Lien, and Nancy M Amato. Roadmap-based flocking for complex environments. In *Proceedings of the Pacific Conference on Computer Graphics and Applications*, pages 104–113. IEEE, 2002.

[BM03] Bobby D Bryant and Risto Miikkulainen. Neuroevolution for adaptive teams. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 3, pages 2194–2201, 2003.

[BMdOB03] Adriana Braun, Soraia Raupp Musse, Luiz Paulo Luna de Oliveira, and Bardo EJ Bodmann. Modeling individual behaviors in crowd simulation. In *Proceedings of the International Conference on Computer Animation and Social Agents*, pages 143–148. IEEE, 2003.

[Bri81] Anne Brindle. Genetic algorithms for function optimization. 1981.

[Bro89] Rodney A Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural computation*, 1(2):253–262, 1989.

[Bro02] Mark Brockington. Level-of-detail ai for a large role-playing game. *AI Game Programming Wisdom*, 1:419–425, 2002.

[BT80] Aharon Ben-Tal. *Characterization of Pareto and lexicographic optimal solutions*. Springer, 1980.

[BT97] Riza C Berkan and Sheldon Trubatch. *Fuzzy System Design Principles*. Wiley-IEEE Press, 1997.

[BYBS10] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

[Cau] Massive brain. `http://www.maa.org/publications/maa-reviews/an-introduction-to-heavy-tailed-and-subexponential-distributions`. Accessed: 2015-04-28.

[CC57] Abraham Charnes and William W Cooper. Management models and industrial applications of linear programming. *Management Science*, 4(1):38–91, 1957.

[CC07] Nicolas Courty and Thomas Corpetti. Crowd motion capture. *Computer Animation and Virtual Worlds*, 18(4-5):361–370, 2007.

[CF99] Kumar Chellapilla and David B Fogel. Evolving neural networks to play checkers without relying on expert knowledge. *IEEE Transactions on Neural Networks*, 10(6):1382–1391, 1999.

[Che04] Stephen Chenney. Flow tiles. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 233–242. Eurographics Association, 2004.

[CL94] Yuan-Lin Chen and C-C Liu. Multiobjective var planning using the goal-attainment method. *Generation, Transmission and Distribution*, 141(3):227–232, 1994.

[CL08] Jen-Yao Chang and Tsai-Yen Li. Simulating virtual crowd with fuzzy logics and motion planning for shape template. In *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, pages 131–136. IEEE, 2008.

[CLL09] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. On-line neuroevolution applied to the open racing car simulator. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2622–2629. IEEE, 2009.

[Coe99] Carlos A Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information systems*, 1(3):269–308, 1999.

[CSM] Crowd simulation movies. `http://www.massivesoftware.com/film.html`. Accessed: 2014-07-29.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[D+91] Lawrence Davis et al. *Handbook of genetic algorithms*, volume 115. Van Nostrand Reinhold New York, 1991.

[DAPB08] Funda Durupinar, Jan Allbeck, Nuria Pelechano, and Norman Badler. Creating crowd variation with the ocean personality model. In *Proceedings of the International joint conference on Autonomous agents and Multiagent systems*, pages 1217–1220. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[DHOO05] Simon Dobbyn, John Hamill, Keith O'Conor, and Carol O'Sullivan. Geopostors: a real-time geometry/impostor crowd rendering system. In *Proceedings of the Symposium on Interactive 3D graphics and games*, pages 95–102. ACM, 2005.

[Dij59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DJ75] Kenneth Alan De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. University of Michigan, Ann Arbor, MI, USA, 1975.

[DLJD00] Dumitru Dumitrescu, Beatrice Lazzerini, Lakhmi C Jain, and Anca Dumitrescu. *Evolutionary computation*, volume 18. CRC press, 2000.

[DM92] Dipankar Dasgupta and Douglas R McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks.*, pages 87–96. IEEE, 1992.

[DPAM02] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[DT07] Kusum Deep and Manoj Thakur. A new crossover operator for real coded genetic algorithms. *Applied Mathematics and Computation*, 188(1):895–911, 2007.

[DYDA12] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[ECS89] Larry J. Eshelman, Richard A. Caruana, and J. David Schaffer. Biases in the crossover landscape. In *Proceedings of the International Conference on Genetic Algorithms*, pages 10–19, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[EK95] Russ C Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995.

[Eng07] Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[ES93] Larry J. Eshelman and J. David Schaffer. Real-coded genetic algorithms and interval-schemata. In *Foundation of Genetic Algorithms 2*, pages 187–202, San Mateo, CA, USA, 1993. Morgan Kaufmann.

[Eys92] Hans J Eysenck. Four ways five factors are not basic. *Personality and individual differences*, 13(6):667–673, 1992.

[FF+93] Carlos M Fonseca, Peter J Fleming, et al. Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In *Proceedings of the International Conference on Genetic Algorithms*, volume 93, pages 416–423, 1993.

[FHHQ04] David B Fogel, Timothy J Hays, SH Hahn, and James Quon. A self-learning evolutionary chess program. *Proceedings of the IEEE*, 92(12):1947–1954, 2004.

[FI99] Minoru Fukui and Yoshihiro Ishibashi. Self-organized phase transitions in cellular automaton models for pedestrians. *Journal of the physical society of Japan*, 68(8):2861–2863, 1999.

[FKMS04] Dario Floreano, Toshifumi Kato, Davide Marocco, and Eric Sauser. Coevolution of active vision and feature selection. *Biological cybernetics*, 90(3):218–228, 2004.

[FMF+94] Dario Floreano, Francesco Mondada, Dario Floreano, Dario Floreano, Francesco Mondada, and Francesco Mondada. *Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot*. ETH-Zürich, 1994.

[Fog93] David B Fogel. Using evolutionary programing to create neural networks that are capable of playing tic-tac-toe. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 875–880. IEEE, 1993.

[For92] Steven Fortune. Voronoi diagrams and delaunay triangulations. *Computing in Euclidean geometry*, 1:193–233, 1992.

[FS98] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[FSN09] Leonardo G Fischer, Renato Silveira, and Luciana Nedel. Gpu accelerated path-planning for multi-agents in virtual environments. In *Proceedings of the Brazilian Symposium on Games and Digital Entertainment*, pages 101–110. IEEE, 2009.

[FTT99] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *Proceedings of ACM SIGGRAPH*, pages 29–38. ACM Press/Addison-Wesley Publishing Co., 1999.

[Gam] Game trees. `https://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html`. Accessed: 2014-12-07.

[GCC+10] Stephen J Guy, Jatin Chhugani, Sean Curtis, Pradeep Dubey, Ming Lin, and Dinesh Manocha. Pledestrians: a least-effort approach to crowd simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 119–128. Eurographics Association, 2010.

[GD91] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996, 1991.

[Ger10] Roland Geraerts. Planning short paths with clearance using explicit corridors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1997–2004. IEEE, 2010.

[GKG96] D Gorinevsky, A Kapitanovsky, and AA Goldenberg. Neural network architecture for trajectory generation and control of automated car parking. *IEEE Transactions on Control Systems Technology*, 4(1):50–56, 1996.

[GKLM11] Stephen J Guy, Sujeong Kim, Ming C Lin, and Dinesh Manocha. Simulating heterogeneous crowd behaviors using personality trait theory. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 43–52. ACM, 2011.

[GM99] Faustino J Gomez and Risto Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *Proceedings of the International joint conference on Artificial Intelligence*, volume 99, pages 1356–1361, 1999.

[GM03a] Faustino J Gomez and Risto Miikkulainen. Active guidance for a finless rocket using neuroevolution. In *Proceedings of the 5th annual conference on Genetic and evolutionary computation*, pages 2084–2095. Springer, 2003.

[GM03b] Faustino John Gomez and Risto Miikkulainen. *Robust non-linear control through neuroevolution*. Computer Science Department, University of Texas at Austin, 2003.

[GO07] Roland Geraerts and Mark H Overmars. The corridor map method: Real-time high-quality path planning. In *Proceedings of the International conference on Robotics and Automation*, pages 1023–1028, 2007.

[GR94] Claudia V Goldman and Jeffrey S Rosenschein. Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence*, pages 408–413, 1994.

[GSM06] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Efficient non-linear control through neuroevolution. In *Machine Learning: ECML 2006*, pages 654–662. Springer, 2006.

[GvdPvdS13] Thomas Geijtenbeek, Michiel van de Panne, and A Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 32(6):206, 2013.

[GWP96] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 81–89. MIT Press, 1996.

[Han05] Nikolaus Hansen. The cma evolution strategy: A tutorial. *Vu le*, 29, 2005.

[HBD02] Serge P Hoogendoorn, Piet HL Bovy, and Winnie Daamen. Microscopic pedestrian wayfinding and dynamics modelling. *Pedestrian and evacuation dynamics*, 123:154, 2002.

[HDK+06] Brad Hiebert, Jubin Dave, Tae-Yong Kim, Ivan Neulander, Hans Rijpkema, and Will Telford. The chronicles of narnia: the lion, the crowds and rhythm and hues author video presentations are available from the citation page. In *ACM SIGGRAPH Course Notes*, page 1. ACM, 2006.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[HES+08] Raia Hadsell, Ayse Erkan, Pierre Sermanet, Marco Scoffier, Urs Muller, and Yann LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 628–633. IEEE, 2008.

[HK04] Nikolaus Hansen and Stefan Kern. Evaluating the cma evolution strategy on multimodal test functions. In *Parallel Problem Solving from Nature*, pages 282–291. Springer, 2004.

[HM95] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[HMI09] Verena Heidrich-Meisner and Christian Igel. Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, 64(4):152–168, 2009.

[HNOC10] Choon Sing Ho, Quang Huy Nguyen, Yew-Soon Ong, and Xianshun Chen. Autonomous multi-agents in flexible flock formation. In *Motion in Games*, pages 375–385. Springer, 2010.

[HNR68] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Hol75] John H Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[Hol92] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[Hol96] Owen E Holland. Multiagent systems: Lessons from social insects and collective robotics. In *Proceedings of the AAAI Spring Symposium on Adaptation, Coevolution and Learning in Multiagent Systems*, pages 57–62, 1996.

[HRBvG95] Barbara Hayes-Roth, Lee Brownston, and Robert van Gent. Multiagent collaboration in directed improvisation. In *Proceedings of the International Conference on Multiagent Systems*, pages 148–154, 1995.

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[HTR93] Robert V Hogg, Elliot A Tanis, and Meda Jagan Mohan Rao. *Probability and statistical inference*, volume 993. Macmillan New York, 1993.

[Hug03] Roger L Hughes. The flow of human crowds. *Annual review of fluid mechanics*, 35(1):169–182, 2003.

[Jac09] David Jacka. *High-Level Control of Agent-based Crowds by means of General Constraints*. PhD thesis, University of Cape Town, 2009.

[JCP+10] Eunjung Ju, Myung Geol Choi, Minji Park, Jehee Lee, Kang Hoon Lee, and Shigeo Takahashi. Morphable crowds. In *ACM Transactions on Graphics*, volume 29, page 140. ACM, 2010.

[JLLW08] Chia-Feng Juang, Chun-Ming Lu, Chiang Lo, and Chi-Yen Wang. Ant colony optimization algorithm for fuzzy controller design and its fpga implementation. *IEEE Transactions on Industrial Electronics*, 55(3):1453–1462, 2008.

[JPVdS01] Wander Jager, Roel Popping, and Hans Van de Sande. Clustering and fighting in two-party crowds: Simulating the approach-avoidance conflict. *Journal of Artificial Societies and Social Simulation*, 4(3):1–18, 2001.

[Ken10] James Kennedy. Particle swarm optimization. In *Encyclopedia of Machine Learning*, pages 760–766. Springer, 2010.

[KGML12] Sujeong Kim, Stephen J Guy, Dinesh Manocha, and Ming C Lin. Interactive simulation of dynamic crowd behaviors using general adaptation syndrome theory. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 55–62. ACM, 2012.

[KGO09] Ioannis Karamouzas, Roland Geraerts, and Mark Overmars. Indicative routes for path planning and crowd simulation. In *Proceedings of the International Conference on Foundations of Digital Games*, pages 113–120. ACM, 2009.

[KLM96] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *arXiv preprint cs/9605103*, 1996.

[KMKWS01] Hubert Klüpfel, Tim Meyer-König, Joachim Wahle, and Michael Schreckenberg. Microscopic simulation of evacuation processes on passenger ships. In *Theory and practical issues on cellular automata*, pages 63–71. Springer, 2001.

[KO04] Arno Kamphuis and Mark H Overmars. Finding paths for coherent groups using clearance. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28. Eurographics Association, 2004.

[KS02] Ansgar Kirchner and Andreas Schadschneider. Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. *Physica A: Statistical Mechanics and its Applications*, 312(1):260–276, 2002.

[KSLO96] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[Lak07] Amit Lakhani. *Multi-agent Systems in Massive*. PhD thesis, MSc Thesis, Bournemouth University, UK, 2007.

[Lap] Laplace distribution. `http://en.wikipedia.org/wiki/Laplace_distribution`. Accessed: 2015-04-30.

[LaV98] Steven M LaValle. Rapidly-exploring random trees a ew tool for path planning. 1998.

[LBLL09] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40, 2009.

[LCF05] Yu-Chi Lai, Stephen Chenney, and ShaoHua Fan. Group motion graphs. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 281–290. ACM, 2005.

[LCHL07] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. Group behavior from video: a data-driven approach to crowd simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 109–118. Eurographics Association, 2007.

[Lip87] Richard P Lippmann. An introduction to computing with neural nets. *IEEE Acoustics, Speech, and Signal Processing Magazine*, 4(2):4–22, 1987.

[Lip89] Richard P Lippmann. Review of neural networks for speech recognition. *Neural computation*, 1(1):1–38, 1989.

[LK06] Manfred Lau and James J Kuffner. Precomputed search trees: planning for interactive goal-driven animation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 299–308. Eurographics Association, 2006.

[LM01] Alex Lubberts and Risto Miikkulainen. Co-evolving a go-playing neural network. In *Proceedings of the 3rd annual conference on Genetic and evolution computation conference*, pages 14–19, 2001.

[LMD+11] Quoc V. Le, Rajat Monga, Matthieu Devin, Greg Corrado, Kai Chen, Marc'Aurelio Ranzato, Jeffrey Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. *CoRR*, abs/1112.6209, 2011.

[LMS01] Helena R Lourenço, Olivier C Martin, and Thomas Stutzle. Iterated local search. *arXiv preprint math/0102188*, 2001.

[LS08] Joel Lehman and Kenneth O Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *Artificial Life*, pages 329–336, 2008.

[Lue03] David P Luebke. *Level of detail for 3D graphics*. Morgan Kaufmann, 2003.

[Masa] An introduction to heavy-tailed and subexponential distributions. `http://iveneverdonethat.com/massive/fuzzy-logic/`. Accessed: 2014-11-21.

[Masb] Massive massive brain. `http://vfx-bex.blogspot.com/`. Accessed: 2014-11-21.

[Masc] Massive software. `http://www.webcitation.org/6UN9aXK0y`. Accessed: 2014-11-26.

[MC68] Donald Michie and RA Chambers. Boxes: An experiment in adaptive control. *Machine intelligence*, 2(2):137–152, 1968.

[MC96] Maja Matarić and Dave Cliff. Challenges in evolving controllers for physical robots. *Robotics and autonomous systems*, 19(1):67–83, 1996.

[MD89] David J Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 89, pages 762–767, 1989.

[MH12] Hirotaka Moriguchi and Shinichi Honiden. Cma-tweann: efficient optimization of neural networks via self-adaptation and seamless augmentation. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation conference*, pages 903–910. ACM, 2012.

[Mic96] Zbigniew Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer, 1996.

[MM94] David E Moriarty and Risto Miikkulainen. Evolving neural networks to focus minimax search. In *AAAI*, pages 1371–1377, 1994.

[MM95] David E Moriarty and Risto Miikkulainen. Discovering complex othello strategies through evolutionary neural networks. *Connection Science*, 7(3-4):195–210, 1995.

[MM96] David E Moriarty and Risto Mikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine learning*, 22(1-3):11–32, 1996.

[MM97] David E Moriarty and Risto Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4):373–399, 1997.

[Mor90] Donald F Morrison. Multivariate statistical methods. 3. *New York, NY. Mc*, 1990.

[MOS09] Ramin Mehran, Alexis Oyama, and Mubarak Shah. Abnormal crowd behavior detection using social force model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 935–942. IEEE, 2009.

[MPT92] Clark McPhail, William T Powers, and Charles W Tucker. Simulating individual and collective action in temporary gatherings. *Social Science Computer Review*, 10(1):1–28, 1992.

[MT93] Michael de la Maza and Bruce Tidor. An analysis of selection procedures with particular attention paid to proportional and boltzmann selection. In *Proceedings of the International Conference on Genetic Algorithms*, pages 124–131. Morgan Kaufmann Publishers Inc., 1993.

[MTSC04] Nadia Magnenat-Thalmann, Hyewon Seo, and Frederic Cordier. Automatic modeling of virtual humans and body clothing. *Journal of Computer Science and Technology*, 19(5):575–584, 2004.

[MUAT05] Soraia Raupp Musse, Branislav Ulicny, Amaury Aubel, and Daniel Thalmann. Groups and crowd simulation. In *ACM SIGGRAPH Course Notes*, page 2. ACM, 2005.

[MVG] Multi-variate gaussian distribution. `http://en.wikipedia.org/wiki/Covariance_matrix`. Accessed: 2015-04-26.

[MW+47] Henry B Mann, Donald R Whitney, et al. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 18(1):50–60, 1947.

[NGCL09] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C Lin. Aggregate dynamics for dense crowd simulation. In *ACM Transactions on Graphics*, volume 28, page 122. ACM, 2009.

[NT93] Hansrudi Noser and Daniel Thalmann. L-system-based behavioral animation. In *Proceedings of Pacific Graphics*, volume 93, pages 133–146. Citeseer, 1993.

[NT96] Hansrudi Noser and Daniel Thalmann. The animation of autonomous actors based on production rules. In *Proceedings of Computer Animation*, pages 47–57. IEEE, 1996.

[NTT92] Hansrudi Noser, Daniel Thalmann, and Russell Turner. Animation based on the interaction of l-systems with vector force fields. In *Visual Computing*, pages 747–761. Springer, 1992.

[OK97] I Ono and S Kobayashi. A Real-Coded Genetic Algorithm for Function Optimization using Unimodal Normal Distribution Crossover. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 246–253. Morgan Kaufmann, 1997.

[OL96] Ibrahim H Osman and Gilbert Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

[OM93] Shigeyuki Okazaki and Satoshi Matsushita. A study of simulation model for pedestrian movement with evacuation and queuing. In *Proceedings of the International Conference on Engineering for Crowd Safety*, pages 271–280, 1993.

[OPOD10] Jan Ondřej, Julien Pettré, Anne-Hélène Olivier, and Stéphane Donikian. A synthetic-vision based steering approach for crowd simulation. In *ACM Transactions on Graphics*, volume 29, page 123. ACM, 2010.

[PAB07] Nuria Pelechano, Jan M Allbeck, and Norman I Badler. Controlling individual agents in high-density crowd simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 99–108. Eurographics Association, 2007.

[PB09] Matt Parker and Bobby D Bryant. Lamarckian neuroevolution for visual control in the quake ii environment. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2630–2637. IEEE, 2009.

[PDJ94] Mitchell A Potter and Kenneth A De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel problem solving from naturePPSN III*, pages 249–257. Springer, 1994.

[PLL98] MV Nagendra Prasad, Victor R Lesser, and Susan E Lander. Learning organizational roles for negotiated search in a multiagent system. *International Journal of Human-Computer Studies*, 48(1):51–67, 1998.

[PMGW04] Mikel D Petty, Frederic D McKenzie, Ryland C Gaskins, and Eric W Weisel. Developing a crowd federate for military simulation. In *Proceedings of the Simulation Interoperability Workshop*, pages 483–493, 2004.

[PP98] João Carlos Figueira Pujol and Riccardo Poli. Evolving the topology and the weights of neural networks using a dual representation. *Applied Intelligence*, 8(1):73–84, 1998.

[PP07] Matt Parker and Gary B Parker. The evolution of multi-layer neural networks for the control of xpilot agents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 232–237. IEEE, 2007.

[PPD07] Sébastien Paris, Julien Pettré, and Stéphane Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. In *Proceedings of the Computer Graphics Forum*, volume 26, pages 665–674. Wiley Online Library, 2007.

[PSY88] Demetri Psaltis, Athanasios Sideris, and Alan Yamamura. A multilayered neural network controller. *IEEE control systems magazine*, 8(2):17–21, 1988.

[PT01] Alan Penn and Alasdair Turner. Space syntax based agent simulation. 2001.

[PVDBC+11] Sachin Patil, Jur Van Den Berg, Sean Curtis, Ming C Lin, and Dinesh Manocha. Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):244–254, 2011.

[QH10] Fasheng Qiu and Xiaolin Hu. Modeling group structures in pedestrian crowd simulation. *Simulation Modelling Practice and Theory*, 18(2):190–205, 2010.

[QSMH03] Matt Quinn, Lincoln Smith, Giles Mayley, and Phil Husbands. Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1811):2321–2343, 2003.

[RER94] Brian J Ritzel, J Wayland Eheart, and S Ranjithan. Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. *Water Resources Research*, 30(5):1589–1603, 1994.

[Rey87] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. *Proceedings of ACM SIGGRAPH*, 21(4):25–34, 1987.

[Rey99] Craig W Reynolds. Steering behaviors for autonomous characters. In *Proceedings of the Game Developers Conference*, volume 1999, pages 763–782, 1999.

[Rey06] Craig Reynolds. Big fast crowds on ps3. In *Proceedings of the ACM SIGGRAPH Symposium on Video Games*, pages 113–121. ACM, 2006.

[rHNG93] Je rey Horn, Nicholas Nafpliotis, and David E Goldberg. Multiobjective optimization using the niched pareto genetic algorithm. *Illinois Genetic Algorithms Lab report*, (93005):61801–2296, 1993.

[RHW88] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 1988.

[RMM98] Norman Richards, David E Moriarty, and Risto Miikkulainen. Evolving neural networks to play go. *Applied Intelligence*, 8(1):85–96, 1998.

[RMT01] Soraia Raupp Musse and Daniel Thalmann. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):152–164, 2001.

[RRM+11] Padmini Rajagopalan, Aditya Rawal, Risto Miikkulainen, Marc A Wiseman, and Kay E Holekamp. The role of reward structure, coordination mechanism and net return in the evolution of cooperation. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 258–265. IEEE, 2011.

[SAR98] Sandip Sen, Neeraj Arora, and Shounak Roychowdhury. Using limited information to enhance group stability. *International Journal of Human-Computer Studies*, 48(1):69–82, 1998.

[SBM05] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. Evolving neural network agents in the nero video game. *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 182–189, 2005.

[SC66] JF Schaefer and RH Cannon. On the control of unstable mechanical systems. In *Proceedings of the Third Congress of the International Federation of Automatic Control*, volume 6, page 1966, 1966.

[SD94] Nidamarthi Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[Sel56] Hans Selye. The stress of life. 1956.

[SGC04] Mankyu Sung, Michael Gleicher, and Stephen Chenney. Scalable behaviors for crowd simulation. In *Proceedings of the Computer Graphics Forum*, volume 23, pages 519–528. Wiley Online Library, 2004.

[She11] Gene Sher. Dxnn: evolving complex organisms in complex environments using a novel tweann system. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 149–150. ACM, 2011.

[Sim94a] Karl Sims. Evolving 3d morphology and behavior by competition. *Artificial life*, 1(4):353–372, 1994.

[Sim94b] Karl Sims. Evolving virtual creatures. In *Proceedings of ACM SIGGRAPH*, pages 15–22. ACM, 1994.

[SKP97] Daniel Svozil, Vladimir Kvasnicka, and Jirí Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[SM02] Kenneth Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[SM04] Kenneth Owen Stanley and Risto P Miikkulainen. *Efficient evolution of neural networks through complexification.* Citeseer, 2004.

[SM08] Jacob Schrum and Risto Miikkulainen. Constructing complex npc behavior via multi-objective neuroevolution. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 8, pages 108–113, 2008.

[SOHTG99] Thorsten Schelhorn, David O'Sullivan, Mordechay Haklay, and Mark Thurstain-Goodwin. Streets: an agent-based pedestrian model. 1999.

[SP95] Rainer Storn and Kenneth Price. *Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces*, volume 3. ICSI Berkeley, 1995.

[SR14] Julian Togelius Sebastian Risi. Neuroevolution in games: State of the art and open challenges. 2014.

[Sti00] G Keith Still. *Crowd dynamics.* PhD thesis, University of Warwick, 2000.

[Sut84] Richard Stuart Sutton. Temporal credit assignment in reinforcement learning. 1984.

[SV00] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[SVR99] Peter Stone, Manuela Veloso, and Patrick Riley. The cmunited-98 champion simulator team. In *RoboCup-98: Robot soccer world cup II*, pages 61–76. Springer, 1999.

[SWL11] Jason Sewall, David Wilkie, and Ming C Lin. Interactive hybrid simulation of large-scale traffic. In *ACM Transactions on Graphics*, volume 30, page 135. ACM, 2011.

[Sys89] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann.

[Tan93] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the International Conference on Machine Learning*, volume 337. Amherst, MA, 1993.

[TBL+07] Julian Togelius, Peter Burrow, Simon M Lucas, et al. Multi-population competitive co-evolution of car racing controllers. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 4043–4050, 2007.

[TCP06] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. In *ACM Transactions on Graphics*, volume 25, pages 1160–1168. ACM, 2006.

[TGLT14] Jie Tan, Yuting Gu, C Karen Liu, and Greg Turk. Learning bicycle stunts. *ACM Transactions on Graphics*, 33(4), 2014.

[TGMY09] Daniel Thalmann, Helena Grillon, Jonathan Maim, and Barbara Yersin. Challenges in crowd simulation. In *Proceedings of the International Conference on CyberWorlds*, number EPFL-CONF-159013, pages 1–12. IEEE Service Center, 445 Hoes Lane, Po Box 1331, Piscataway, Nj 08855-1331 Usa, 2009.

[Tha07] Daniel Thalmann. *Crowd simulation*. Wiley Online Library, 2007.

[THH00] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Microscopic simulation of congested traffic. In *Traffic and granular flow*, pages 365–376. Springer, 2000.

[THL+04] Daniel Thalmann, Christophe Hery, Seth Lippman, Hiromi Ono, Stephen Regelous, and Douglas Sutton. Crowd and group animation. In *ACM SIGGRAPH Course Notes*, page 34. ACM, 2004.

[TKKS09] Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jürgen Schmidhuber. Super mario evolution. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 156–161. IEEE, 2009.

[TL05] Julian Togelius and Simon M Lucas. Evolving controllers for simulated car racing. In *Proceedings of the IEE Congress on Evolutionary Computation*, volume 2, pages 1906–1913. IEEE, 2005.

[TL06] Julian Togelius and Simon M Lucas. Evolving robust and specialized car racing skills. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1187–1194. IEEE, 2006.

[TLC02] Franco Tecchia, Celine Loscos, and Yiorgos Chrysanthou. Visualizing crowds in real-time. In *Proceedings of the Computer Graphics Forum*, volume 21, pages 753–765. Wiley Online Library, 2002.

[TP02] Alasdair Turner and Alan Penn. Encoding natural movement as an agent-based system: an investigation into human pedestrian behaviour in the built environment. *Environment and Planning B*, 29(4):473–490, 2002.

[TSM99] Charles W Tucker, David Schweingruber, and Clark McPhail. Simulating arcs and rings in gatherings. *International journal of human-computer studies*, 50(6):581–588, 1999.

[TT94] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of ACM SIGGRAPH*, pages 43–50. ACM, 1994.

[TT12] Jason M Traish and James R Tulip. Towards adaptive online rts ai with neat. In *Proceedings of the IEEE conference on Computational Intelligence and Games*, pages 430–437. IEEE, 2012.

[UCT04] Branislav Ulicny, Pablo de Heras Ciechomski, and Daniel Thalmann. Crowdbrush: interactive authoring of real-time crowd scenes. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 243–252. Eurographics Association, 2004.

[VdBE04] Frans Van den Bergh and Andries Petrus Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, 2004.

[VdBLM08] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

[WGN14] Sunrise Wang, James Gain, and Geoff Nitschke. Comparing crossover operators in neuro-evolution with crowd simulations. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2298–2305. IEEE, 2014.

[Wie91] Alexis P Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 667–673. IEEE, 1991.

[Wig96] Jerry S Wiggins. *The five-factor model of personality: Theoretical perspectives*. Guilford Press, 1996.

[WKF09] Markus Waibel, Laurent Keller, and Dario Floreano. Genetic team composition and level of selection in the evolution of cooperation. *IEEE Transactions on Evolutionary Computation*, 13(3):648–660, 2009.

[Woo00] Ronald F Woodaman. Agent-based simulation of military operations other than war small unit combat. Technical report, DTIC Document, 2000.

[Wri91] Alden H. Wright. Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann, 1991.

[WWW65] James Hardy Wilkinson, James Hardy Wilkinson, and James Hardy Wilkinson. *The algebraic eigenvalue problem*, volume 87. Clarendon Press Oxford, 1965.

[XOR] G5aiai - introduction to artificial intelligence. `http://www.cs.nott.ac.uk/~gxk/courses/g5aiai/006neuralnetworks/neural-networks.htm`. Accessed: 2014-12-07.

[YCP+08] Hengchin Yeh, Sean Curtis, Sachin Patil, Jur van den Berg, Dinesh Manocha, and Ming Lin. Composite agents. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 39–47. Eurographics Association, 2008.

[Yeg09] B Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[YM09] Chern Han Yong and Risto Miikkulainen. Coevolution of role-based cooperation in multiagent systems. *IEEE Transactions on Autonomous Mental Development*, 1(3):170–186, 2009.

[YMPT09] Barbara Yersin, Jonathan Maïm, Julien Pettré, and Daniel Thalmann. Crowd patches: populating large-scale virtual environments for real-time applications. In *Proceedings of the Symposium on Interactive 3D graphics and Games*, pages 207–214. ACM, 2009.

[zhe] Adidas zheng zhi advertisement poster. `http://za.adforum.com/creative-work/ad/player/12653140`. Accessed: 2015-04-20.

[Zip49] George Kingsley Zipf. Human behavior and the principle of least effort. 1949.

Appendices

Appendix A

Storing and Loading ANNs

ANNs are stored and loaded from XMLs. These XMLs can be used to either only define structure, or used to define the exact weightings of an ANN.

An example of an XML that only defines structure is:

```
<?xml version="1.0" ?>
<NeuralNetworks>
  <NeuralNetwork>
    <Neuron ID = "1" Type = "Input"/>
    <Neuron ID = "2" Type = "Input"/>
    <Neuron ID = "3" Type = "Hidden" Activation = "Sigmoid">
      <Predecessors>
        <Predecessor ID = "1"/>
        <Predecessor ID = "2"/>
      </Predecessors>
      <Weights Distribution = "Uniform" Min = "-1" Max = "1"/>
    </Neuron>

    <Neuron ID = "4" Type = "Hidden" Activation = "Sigmoid">
      <Predecessors>
        <Predecessor ID = "1"/>
          <Predecessor ID = "2"/>
        </Predecessors>
        <Weights Distribution = "Uniform" Min = "-1" Max = "1"/>
    </Neuron>

    <Neuron ID = "5" Type = "Output" Activation = "Sigmoid">
      <Predecessors>
        <Predecessor ID = "3"/>
          <Predecessor ID = "4"/>
      </Predecessors>
      <Weights Distribution = "Uniform" Min = "-1" Max = "1"/>
    </Neuron>
  </NeuralNetwork>
</NeuralNetworks>
```

Distribution signifies what distribution the weights are sampled from, and *Activation* signifies what activation function the neuron uses.

An example of an XML that defines both weights and structure of an ANN is:

```
<?xml version="1.0" ?>
<NeuralNetworks>
  <NeuralNetwork>
    <Neuron ID = "1" Type = "Input"/>
    <Neuron ID = "2" Type = "Input"/>
    <Neuron ID = "3" Type = "Hidden" Activation = "Sigmoid">
      <Predecessors>
        <Predecessor ID = "1"/>
        <Predecessor ID = "2"/>
      </Predecessors>
      <Weights Distribution = "Fixed">
        <Weight Value = "0.3">
        <Weight Value = "-0.4">
        <Weight Value = "-0.1">
      </Weights>
    </Neuron>

    <Neuron ID = "4" Type = "Hidden" Activation = "Sigmoid">
      <Predecessors>
        <Predecessor ID = "1"/>
          <Predecessor ID = "2"/>
      </Predecessors>
      <Weights Distribution = "Fixed">
        <Weight Value = "-0.3">
        <Weight Value = "-0.9">
        <Weight Value = "1">
      </Weights>
    </Neuron>

    <Neuron ID = "5" Type = "Output" Activation = "Sigmoid">
      <Predecessors>
        <Predecessor ID = "3"/>
          <Predecessor ID = "4"/>
      </Predecessors>
      <Weights Distribution = "Fixed">
        <Weight Value = "0.5">
        <Weight Value = "0.2">
        <Weight Value = "-0.4">
      </Weights>
    </Neuron>
  </NeuralNetwork>
</NeuralNetworks>
```

There reason there is 1 more weight than predecessors is because it acts as the bias weight, which is implicit to the system.

Appendix B

System Information

Language: C++
Compiler: MSVC9

Libraries:

- **Boost**[1] - for multi-threading, tuples, lexical casting, random number generation, and various other general functionalities.
- **PugiXML**[2] - for loading and storing of XML files.
- **BulletPhysics**[3] - for collision detection, force calculations, and ray-casting.
- **Eigen**[4] - for linear algebra functions.
- **Ogre3D**[5] - for rendering and loading 3D assets.
- **Microsoft MPI**[6] - for distributing the system.
- **Blender**[7] - for modeling agents and their environments.

[1] http://www.boost.org/
[2] http://pugixml.org/
[3] http://bulletphysics.org/wordpress/
[4] http://eigen.tuxfamily.org/
[5] http://www.ogre3d.org/
[6] http://msdn.microsoft.com/en-us/library/bb524831(v=vs.85).aspx
[7] http://www.blender.org/

Appendix C

NE algorithm parameters

C.1 CNE

Parameters	Value
Population size	100
Selection	Linear Rank
Mutation	Gaussian 0.2
Crossover	Laplace
Crossover Probability	0.8
Elitism	5%

C.2 NEAT

Parameters	Value
Population size	100
Selection	Linear Rank
Elitism	5%
c_1	1
c_2	1
c_3	0.4
Compatibility Threshold	1
Weight mutation	Gaussian 0.2
Add node probability	0.03
Add connection probability	0.05
Mutate only probability	0.25
Interspecies crossover probability	0.001
Crossover only probability	0.2
Crossover by choosing probability	0.6
Crossover by averaging probability	0.4
Maximum stagnation	15

C.3 ESP

Parameters	Value
Sub-population Size	50
Evaluations per neuron	5
Burst threshold	20
Crossover	Laplace
Crossover Probability	0.8
Mutation	Gaussian 0.2
Selection	Linear Rank
Elitism	10%

C.4 CMA-ES

Parameters	Value
Initial step size	0.2