2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Comparing crossover operators in Neuro-Evolution with Crowd Simulations

Sunrise Wang James Gain Geoff Nitschke

***Abstract*— Crowd simulations are a set techniques used to control groups of agents and are exemplified by scenes from movies such as The Lord of the Rings and Inception. A problem which all crowd simulation techniques suffer from is the balance between control of the crowd behaviour and the autonomy of the agents. One possible solution to this problem is to use Neuro-Evolution (NE) to evolve the agent models so that the agents behave realistically and the emergent crowd behaviour is controllable. Since this is not an application area which has been investigated much, it is unknown which NE parameters and operators work well. This paper attempts to address this by comparing the performance of a set of crossover operators with a range of probabilities in three simulations: Car Racing, Mouse Bridge Crossing, and a War-Robot Battle. Overall it was found that Laplace crossover worked the best across all our simulations.**

I. Introduction

CROWD SIMULATIONS are a set of techniques used to simulate groups of agents within virtual environments. One of their most well known applications is to the film *The Lord of the Rings*, where battle scenes were created using the crowd simulation software *Massive*[1]. Other application areas for crowd simulations include the planning of evacuation routes, architectural and urban design, and video games.

A considerable amount of research has been undertaken on crowd simulations, giving rise to two paradigms: *Microscopic* (bottom-up) [21], [12], [26] and *Macroscopic* [4], [25], [15] (top-down). *Microscopic* techniques simulate the behaviours of individual agents by providing them with local information and rules, with the aim of creating emergent crowd behaviour. *Macroscopic* techniques aim to control global characteristics of a crowd and agents are updated in order to reflect such.

A requirement in industries such as film is that crowds be both believable and controllable. Macroscopic approaches, despite providing easier control over their Microscopic counterparts, are infeasible as the agents lack believability due to their overly homogeneous behaviours. Microscopic approaches, on the other hand, require users to perform the difficult and time-consuming task of adjusting the local behaviours of agents until the desired global behaviour emerges.

One possible solution is using an optimisation algorithm [8] to find the ideal local models for the agents within a Microscopic crowd. In this paper we investigate one such technique, namely Neuro-Evolution (NE). NE is advantageous compared to other optimisation methods because one does not need to define the initial behavioural models of the agents as Artificial Neural-Networks (ANNs) replace them.

Operators and parameters used by genetic algorithms are often problem dependent. Since controlling emergent crowd behaviour using NE has, to our knowledge, not been thoroughly explored before, it is important to determine which work well within this new context. Since the scope of thoroughly investigating all NE parameters is very large, we are, for the purposes of this paper, narrowing our investigation to focus only on crossover operators and probabilities. We achieve this by fixing other operators and parameters to values predetermined during preliminary tests. We use a testbed of three simulations (car racing, mouse bridge crossing, war-robot battle) in conjunction with a cooperative NE algorithm: Enforced Sub-populations (ESP) [11], which was found to provide more task performance scalability compared to non-cooperative approaches. It was found that Laplace crossover [7] performed the best across all simulations, with Unimodal Normal Distribution Crossover (UNDX) [20] and Arithmetic crossover [19] performing the worst.

The rest of the paper is structured as follows: Section 2 provides context and related works, Section 3 describes how we use NE to control the crowds, as well as the various agent and crowd simulation types used, Section 4 describes our experimentation method, Section 5 presents our findings as well as a brief discussion, Section 6 concludes the paper.

II. Background and Related Work

A. Crowd Simulations

Anderson *et al.* [2] propose a macroscopic technique which allowed hard constraints on a bird flock's agent positions at specified time intervals whilst still allowing for seemingly realistic behaviours. It achieves this by calculating the viable paths of the constrained model, and then choosing the path which most closely resembles an unconstrained model by using a wander element. This technique is interesting in that it allows for hard constraints on the crowd simulation while the agents still appear to act autonomously. This approach unfortunately only deals with the positioning and movement of agents, and has only been tested with bird flocks. Whereas, we are interested in controlling additional behaviours and agent types.

The crowd simulation system *Massive* uses a fuzzy logic system, where users are able to create fuzzy controllers in order to define an agent's behaviour. Jacka [16] extended this by using a microscopic approach where he optimised

James Gain, Geoff Nitschke, and Sunrise Wang are with the Department of Computer Science, University of Cape Town, Cape Town, South Africa (email: {jgain, gnitschke}@cs.uct.ac.za, sunrisewng@gmail.com).

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

the membership functions of the fuzzy nodes with Particle Swarm Optimisation [17] in order for the crowd simulation to reflect the desired behaviours. Jacka's method of evaluating the fitness of a crowd simulation is very similar to ours, where a crowd simulation is run fully and its satisfaction of the various desired behaviours is combined into a sum of weighted values.

Yong and Miikkulainen [29] propose an extension of ESP termed Multi-Agent ESP in order to evolve multiple predator agents to catch a single prey agent in a predator-prey environment. Using Multi-Agent ESP, they evaluate the level of cooperation achieved between the predator agents given various levels of communication. Although their approach is similar to ours in that we also evolve and share fitness values between multiple ANNs, it differs in that they use a distinct ANN for each agent while we use a single ANN across a group of agents. This allows our method to scale better to different crowd sizes.

Schrum and Miikkulainen [22] evolved monsters in a multi-objective landscape in order to train them to find a trade-off between dealing maximal, and receiving minimal damage from the player. Their approach of evaluating fitness differs from ours in that we combine the multiple objectives into a single weighted objective function whereas they used pareto-based multi-objective optimisation. They show that using pareto-based multi-objective optimisation leads to some interesting and realistic agent behaviours and thus may be a future direction for our research to take. They also borrow ideas from NEAT [23] as they evolve both the structure and the weights of the ANN which may be another direction which our system may be extended in. They however did note that using crossover and speciation resulted in an overly homogeneous population.

Bryant and Miikkulainen [3] used NE to evolve heterogeneous behaviours for agents controlled by a single ANN in the video game *Legion-I*. They showed that agents could learn to perform different tasks under different conditions, as long as it contributed to the overall goal. This is highly relevant to our work as it shows that we are able to control a group of agents with a single ANN while still achieving emergent behaviour.

B. Enforced Sub-populations (ESP)

ESP is a cooperative NE method proposed by Gomez and Miikkulainen [11].

ESP differs from NE techniques which evolve full neural-networks in that it recognises that an ANN can be separated into its individual neurons. Therefore, instead of having a population of neural-networks, it instead has n sub-populations of neurons, where n is the amount of neurons within a single neural-network. Each sub-population thus represents a single neuron within the network and recombination is performed only with other neurons in the same sub-population.

In order to evaluate a neuron's fitness, neurons are randomly selected from each sub-population and used to construct an ANN, which is then evaluated by a fitness function in the simulator. The fitness of this ANN is then shared across all the participating neurons. Each neuron is evaluated multiple times so that its fitness is less noisy, as a good neuron can be assigned a poor fitness if other neurons within the ANN are poor.

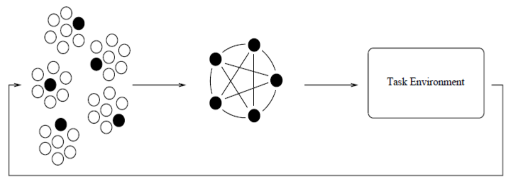

Fig. 1: The general idea of ESP (source [11]). The ANN is created by selecting a random neuron from each sub-population which is represented here by the clusters of circles. Evaluation is performed on this ANN and the fitness is shared across the participating neurons.

In order to maintain genetic diversity within the sub-populations, delta-coding, proposed by Whitley *et al.* [27], is used. Delta-coding searches for the best modifications of the current best solution. It achieves this by using delta-chromosomes, which contain values that represent differences from the current best solution. These delta-chromosomes are then evaluated by adding their values to the current best solution, and are selected for reproduction if they improve it. Although delta-coding is primarily used in the original paper [11] to deal with incremental evolution, we have found that it also improves the fitness values for our simulations. In our system, we use delta-coding after the best solution has not improved after a set number of generations.

C. Crossover Operators

Our chosen operators are described in this section. They were selected because they are well established [13] and are indicative of the various types of crossover operators used in real-coded genetic algorithms. In the equations of the various operators used, we use y to represent offspring and x to represent parents, $U(\alpha, \beta)$ to represent a random number sampled from a uniform distribution with the range [α; β], and $Uint(\alpha, \beta)$ to represent a random integer sampled from a uniform distribution with the range [α; β].

Arithmetic crossover is a multi-parent crossover which produces an offspring as the weighted average of n parents [19]. An offspring is generated from n parents using:

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

where w_k is a random number between 0 and 1 and $\sum_{k=1}^{n} w_k = 1$.

Eshelman and Schaffer [10] propose the *Blend crossover* (BLX-α) where an offspring is generated from two parents using the equations:

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

where $w_j = (1 + 2\alpha)U(0, 1) - \alpha$. Eshelman and schaffer recommended that $\alpha = 0.5$.

The *Heuristic crossover* operator generates one offspring from two parents using the equation [28]

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

where the fitness of x_{1j} is better or equal to the fitness of x_{2j}.

Deep and Thakur [7] propose the *Laplace crossover*, a parent-centric crossover operator where two offspring are generated from two parents using the equations:

$$y_1 = x_1 + \beta|x_1 - x_2|$$

$$y_2 = x_2 + \beta|x_1 - x_2|$$

where

$$\beta = \begin{cases} a + b\ log(2u) & \text{if } u <= 0.5 \\ a - b\ log(2 - 2u) & \text{if } u > 0.5 \end{cases}$$

with $u = U(0,1)$, 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.

Uniform crossover [24] was originally used for reproduction of discrete string representations. Given two parents, an offspring is generated using the equation:

$$y_{ij} = \begin{cases} x_{1j} & \text{if } r_j = 0 \\ x_{2j} & \text{if } r_j = 1 \end{cases}$$

where $r_j = Uint(0,1)$ and is sampled for every index of the chromosome

One-point crossover [14], used originally for discrete-coded genetic algorithms, produces a single offspring from two parents using the equation:

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

where $r = Uint(1,n)$, with n being the dimensionality of the search space

Two-point crossover [9] [5] is another operator for discrete-coded genetic algorithms. A single offspring is produced from two parents using the equation:

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

where $r_1 = Uint(1,n)$ and $r_2 = Uint(1,n)$ with n being the dimensionality of the search space and $r_2 >= r_1$

Deb and Agrawal [1] propose the *simulated binary crossover* (SBX), which simulates the behaviour of one-point crossover for bit string representations. Two offspring are generated from two parents using the equations:

$$y_{1j} = 0.5((1 + w_j)x_{1j} + (1 - w_j)x_{2j}) \text{ and}$$

$$y_{2j} = 0.5((1 - w_j)x_{1j} + (1 + w_j)x_{2j})$$

where

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

and $r_j = U(0,1)$ and $\eta > 0$ is the distribution index and determines how close to the parents the offspring are generated. Deb and Agrawal recommended η = 1 as a setting that generally worked well.

Ono and Kobayashi [20] propose a three-parent center-of-mass operator termed the *Unimodal Normal Distribution Crossover* (UNDX). Offspring can be generated using the equation:

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

where x^p is the midpoint between x_1 and x_2, d is the difference vector $x_1 - x_2$, n is the dimension of the search space, and D is the distance from x_3 to its projection on d. e_i are the orthogonal basis vectors spanning the subspace orthogonal to the vector space defined by d, $\xi = N(0, \sigma_\xi^2)$, $\eta_i = N(0, \sigma_\eta^2)$ with $N(0, \sigma^2)$ being a random number sampled from a gaussian distribution with a mean of 0 and a variance of σ^2. Ono and Kobayashi suggested that $\sigma_\eta = \frac{0.35}{n}$ and $\sigma_\xi = \frac{1}{2}$.

Deb *et al.* [6] proposed the *Parent Centric Crossover* (PCX), a multi-parent operator for real-coded genetic algorithms which generates offspring around the parents. Given μ parents, offspring can be generated with the equation:

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

where x_p is a random parent (the female) chosen for each offspring, $d^{(p)}$ is the direction vector $x_p - g$ with g being the mean vector of the μ parents, D being the average of distances of all the non-female parents onto $d^{(p)}$, e^i being the $\mu - 1$ orthonormal bases that span the subspace orthogonal to $d^{(p)}$, w_ζ and w_μ being zero-mean Gaussian distributed variables with variance of σ_μ^2 and σ_η^2 respectively.

III. Method

There are generally two popular approaches for deploying ANNs to agents in virtual environments. A *homogeneous* approach where all the agents within the virtual environment use a single ANN to determine their behaviour, and a *het-*

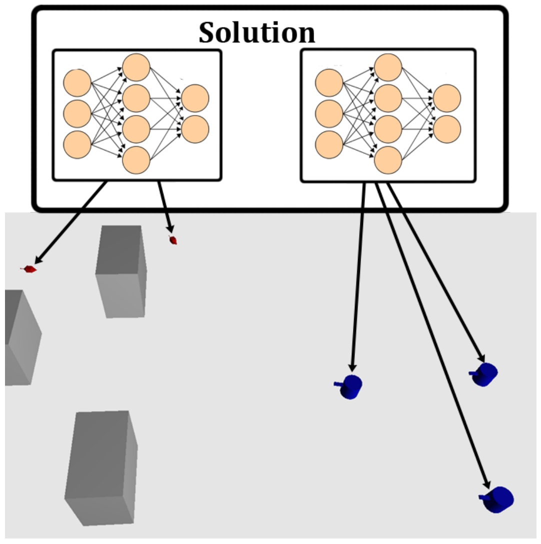

Fig. 2: A crowd simulation where mice are tasked to escape war robots. A solution for such a simulation will consist of two ANNs, one controlling the mouse agents and another controlling the war robot agents, as the mice and war robots have differing behaviours.

erogeneous approach where each agent has its own ANN controller. The heterogeneous approach is infeasible in the context of crowd simulations since there are often hundreds if not thousands of agents within a crowd, which leads to scalability problems. The homogeneous approach is, however, also not desirable as often one wants agents to exhibit different personalities and behaviours (for example, agents assaulting a city in a battle simulation should exhibit much more aggressive behaviours compared to the agents defending it). We thus adopt a semi-homogeneous approach where a set of ANNs are used by agents to determine their behaviour within a simulation. Agents within the simulation are divided into groups determined by their desired behaviours. Each group is assigned an ANN which the agents in the group use in order to determine their behaviour (as seen in figure 2). We found that this approach provides a midpoint between the scalability issues of the heterogeneous approach, and the overly homogeneous agent behaviours of the homogeneous approach.

Figure 3 shows a diagram of our method. Each sub-population of neurons in ESP is assigned to evolve the weights for a given neuron in a given ANN. After the ANNs are constructed via selecting a neuron from each sub-population, they are used as the agent controllers for a given simulation task. The performance of the agents within this simulation is then evaluated by the fitness function, and is then passed back to the ESP algorithm where the fitness is propagated across all involved neurons. Once the training algorithm has terminated, the fittest set of ANNs are rendered in the context of a given simulation.

We implemented three simulations - Car Racing, Mouse Bridge Crossing, and War-robot Battle - to test the 10 chosen

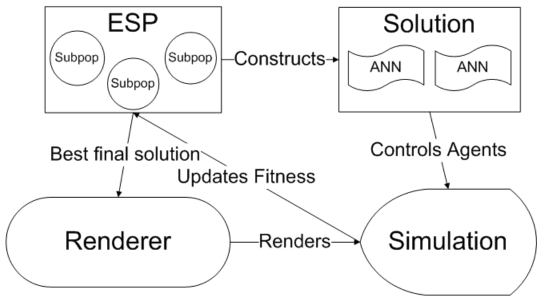

Fig. 3: Our method. A solution is provided to a simulation which is then run in order to obtain a fitness. The fitness is then propagated back to the training algorithm. Once training has finished, the solution is given to the renderer, which then renders the corresponding simulation with the given solution.

crossover operators. While these simulations are by no means exhaustive of the different types of crowd simulations in use in industry, they do provide a sufficiently diverse range of environments and tasks for our tests.

A. Training

1) NE algorithm and parameters: The NE algorithm used for our tests is ESP. We chose this algorithm over approaches which evolved entire networks because it allowed for smaller population sizes caused by fewer dimensions per sub-population. This is very important in the context of controlling crowd simulations since fitness evaluations are computationally expensive. An additional benefit to ESP is that it allows for more scalability in terms of ANN structure compared to full networks, as adding hidden nodes does not increase the dimensionality of the chromosomes, allowing for crossover parameters to remain the same. One should, however, increase the number of evaluations per chromosome when sub-populations are added as the fitness values obtained become less accurate.

2) Fitness Evaluation: Control of the emergent crowd behaviours is achieved through fitness evaluation where the desired behaviours and goals of the crowd are specified as a set of weighted objectives, which are evaluated and combined after a full run of the simulation. One problem with this approach is that the weights strongly influence the quality of the final scene, as poor weighting results in agents learning incorrect behaviours. Although we obtained the weights through trial and error for the purposes of this paper, it is desirable to either develop a method which eliminates the need for these weights or to use pareto-based multi-objective optimisation instead.

B. Simulations

All the agents used in our simulations are two dimensional (only rotates around y-axis and moves only on the x-z plane). Although one of the main contributions of ESP is that it allows for evolution of Recurrent Neural-Networks, we decided to use Feed Forward Neural Networks (FFNNs) instead as there is no need for our agents to remember previously performed actions. These FFNNs determine both the linear acceleration and the torque of our agents. The ANN structure for the

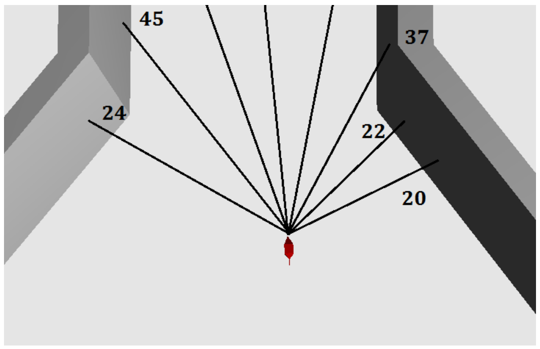

Fig. 4: An agent's vision is simulated by casting rays, and then using the distance to the closest intersected object. These distance values are then passed through to an ANN in order to determine the agent's behaviour.

three types of agents are identical with the exception of the input nodes. Preliminary tests suggested that six hidden nodes worked relatively well.

One type of local information provided to our agents is vision. The way we simulate vision is by casting rays, and returning the distance to the closest obstacle which intersects that ray as shown in figure 4.

1) Car Racing: In this simulation, four car agents are tasked to reach the end of a racetrack before the end of the simulation.

The car moves along its direction of orientation, with its velocity and orientation determined by the ANN which applies both torque and acceleration. The acceleration can be negative, however the agent's velocity cannot be less than zero. The Car agent's ANN structure uses fourteen inputs. These inputs consist of the closest collision distances of rays cast in eight directions distributed uniformly around the agent, the velocity of the agent, the position of the agent's current goal (as way-points are used to help the agent navigate around the racetrack), and the position of the current agent.

The aim is to emulate a race-like scenario where cars have to navigate along a racetrack with multiple corners. The objectives for the agents are to reach the end of the race-track before the simulation ends, and to avoid collisions with other agents. The fitness of a solution is calculated as $c+2d$ where c is the number of collisions between all the agents throughout the simulation, and d is the distance of all the agents to the final goal.

2) Mouse Bridge Crossing: Thirty mouse agents are tasked to cross a bridge before the specified time limit in this simulation.

The Mouse agent behaves very similarly to the car agent in that it moves according to its orientation. It, however, has an additional sensor that detects whether or not there is an obstacle directly in front of it and this determines whether or not it should stop. The Mouse agent provides 16 inputs to its ANN. The inputs consist of the closest collision distances of 8 rays cast in a cone in front of the agent (rather than the radial casting of both the car and war-robot agents), the position of the agent, the velocity of the agent, and the two end-points of the line-segment the agent has to cross.

This tests the ability of agents to learn to navigate through a bottle-neck whilst maintaining a desirable pace. The objectives provided to the agents are that they must cross the bridge and avoid collisions with other agents. The fitness of a solution is calculated as $c+d$ where c is the number of collisions between all the agents throughout the simulation, and d is the distance of all the agents which have not crossed the bridge yet to the end of the bridge.

3) War-robot Battle: In this simulation, two groups of forty War-robot agents are initialised at opposing ends of a map. One group is located behind a city, whereas the other group is initialised in an open field. The two groups of agents, which are controlled by different ANNs, are then tasked to fight each other with the aim being that there should be a specific amount of agents left on both sides by the end of the simulation.

The War-robot agent's motor skills are identical to the Car agent in that it moves forwards with the ANN controlling both the acceleration and torque of the agent. The War-robot agent has the additional behaviour that it will shoot enemies in front of it if they are within shooting range. There are 20 inputs for the War-robot agent, consisting of the closest collision distances as well as the closest collision type flags (-1 for enemy, 0 for environment obstacle, 1 for ally) for rays shot in 8 directions uniformly around the agent, as well as the position and velocity of the agent.

This simulation tests how well NE can control the outcomes of battle-based crowd simulations. The objectives provided are that the agents should avoid collisions with other agents, and that there should be between 9 and 11 agents left on each side after the simulation ends. The fitness of a solution is calculated as $c + 20p$ where c is the number of collisions between the agents, and p is the difference between the desired and actual population sizes. An especially large weight was regarded for p as the fitness contribution for p is typically very small with the maximum amount being 58, thus if a small weight is used, the agents would learn to stand still as that is the optimal behaviour for avoiding collisions.

IV. Experiment

We ran 30 tests for each combination of the 10 crossover operators and 5 crossover probabilities (0.2, 0.4, 0.6, 0.8, and 1) on each of the 3 simulations, with agents given random starting positions at the start of each run. For the n-parent crossover operators, the minimum number of parents was chosen. The rest of the GA operators used for our training are as follows:

- Gaussian mutation with 0.2 standard deviation and 0.02 probability
- Rank-based parent selection with linearly increasing probabilities
- Elitism with 10% population size
- ESP sub-population size of 20
- 3 fitness evaluations per neuron
- Search space of [-1, 1] on all dimensions

- Delta-coding used if the best solution has not improved after 40 generations.
- Algorithm terminates after 200 generations, or if a solution with 0 (we aim to minimize the fitness function) fitness is found

In order to select the mutation and selection operators, we compared some well known operators within the field and selected the ones which appeared to perform the best. Elitism is used as we found that the algorithm converges much more slowly without it. We found that a 10% elitism allowed the algorithm to converge faster but not prematurely most of the times. The ESP sub-population size, fitness evaluations per neuron, and the delta-coding interval were chosen to provide an acceptable midpoint between execution speed and performance. 200 generations was chosen as the stopping point because we observed that the algorithm improved little after this point. The search-space was limited to [-1, 1] because we used the sigmoid activation function for our neurons.

The fitness functions for the various simulations can be found in sections III-B.1, III-B.2, and III-B.3. The fitness ranges for these simulations are as follows: Car Racing = [0, 2400]; Mouse Bridge Crossing = [0, 10500]; War-Robot Battle = [0, 25160].

Although a fitness of 0 is desired, it is not always achievable for simulations with a larger amount of agents. However, the fitness does not have to be 0 in order for the crowd to behave as expected. We found that as long as the fitness values are below the following amounts that the crowd will most likely behave desirably: Car Racing = 10; Mouse Bridge Crossing = 300; War Robot Battle = 100.

The system was implemented in C++ and compiled using the MSVC9 compiler. Ogre 1.8.1[2] was used for the rendering of the simulations. Bullet 2.81[3] was used for the ray-casting and the collision detection. PugiXML[4] was used for the loading and saving of the neural-networks to XML files. The boost 1.51[5] MT19937 implementation of the Mersenne Twister was used for random number generation. The source code and the full results can be found at an online repository[6].

V. Results and Discussion

In order to compare the performance of the various crossover operators and probabilities, we compare the mean fitness values achieved by each combination of crossover operator and probability, and perform the Mann-Whitney U test [18] between these and the operator which achieved the best mean fitness for each simulation (for example, each operator was compared to UNDX 0.4 in the car-racing simulation). We view a p-value of under 0.05 with a confidence interval of 95% as being statistically significant.

Figure 5 shows the mean fitness values obtained by each operator in the car racing simulation at 200 generations. Overall UNDX 0.4 achieved the lowest mean. Despite this, UNDX 0.4 is only statistically significantly better than about a fifth of the other operators. As seen in table I, it is not significantly better than the worst performing operator (Onepoint 0.8). The reason for this is that many of the operators achieved poor mean fitness values for this simulation caused by being stuck in local minima with fitness values which deviate greatly from the fitness values usually achieved. Overall it would appear that most operators were comparable for this particular simulation.

Simulation	Operator	Mean	σ	p-value
Car race	UNDX 0.4	4.971	5.151	N/A
Car race	Laplace 0.6	5.54355	8.127547612	0.888
Car race	BLX-α 1 0.6	6.811	7.038	0.425
Car race	Onepoint 0.8	40.521	97.389	0.105
Car race	BLX-α 0.8	32.253	82.412	0.009
Car race	Onepoint 0.6	28.667	79.339	0.145
MBC	Laplace 0.8	184.094	75.263	N/A
MBC	BLX-α 0.8	190.667	77.784	0.906
MBC	Laplace 1	203.361	93.849	0.549
MBC	UNDX 0.8	490.845	171.194	2.05E-10
MBC	UNDX 1	478.228	255.056	2.1E-08
MBC	Heuristic 1	413.984	61.722	1.15E-10
WRB	Laplace 0.6	17.366	12.516	N/A
WRB	BLX-α 0.6	19.566	13.783	0.589
WRB	BLX-α 0.4	20	19.874	0.882
WRB	Arithmetic 0.8	80.066	95.282	1.58E-05
WRB	UNDX 0.8	82.8667	109.375	4.27E-06
WRB	Arithmetic 1	79.067	98.424	2.132E-05

TABLE I: The mean, standard deviations, and p-values for some of the best and worst operators in each simulation. War-Robot Battle is abbreviated to WRB and Mouse Bridge Crossing is abbreviated to MBC. Statistical tests compared (for a given simulation) each operator with the operator yielding the highest mean task performance (N/A in p-value column).

The mean fitness values obtained by each operator within the mouse bridge crossing simulation at 200 generations are shown in figure 6. Laplace 0.8 achieved the best mean, and also performed significantly better than most of the other operators. Heuristic 0.4 and BLX-α 0.8 also seemed to work very well for this simulation. However, Heuristic crossover's performance deteriorated rapidly as the crossover probabilities were increased. A point to note is that despite UNDX achieving the best mean fitness in the car racing simulation, it performs the worst in both the mouse bridge crossing simulation and the war-robot simulation. This shows that the performance of UNDX deteriorates when dealing with larger groups of agents which in turn increases the difficulty of achieving a high task performance. An explanation for this deterioration is that we are using only the minimum number of parents, and using an optimal amount of parents may lead to better performance.

In figure 7 for the war-robot battle, Laplace 0.6 achieved the best mean fitness, whereas UNDX 0.8 achieved the worst. Despite Laplace crossover consistently achieving very good mean fitness values for all simulations, it should however be noted that BLX-α shows comparable results.

Another observation garnered from our results is that crossover operators originally intended for discrete-coded

[2]http://www.ogre3d.org/
[3]http://bulletphysics.org/wordpress/
[4]https://code.google.com/p/pugixml/
[5]http://www.boost.org/users/history/version_1_51_0.html
[6]https://bitbucket.org/igorawratu/neuroevolution-crowdsim

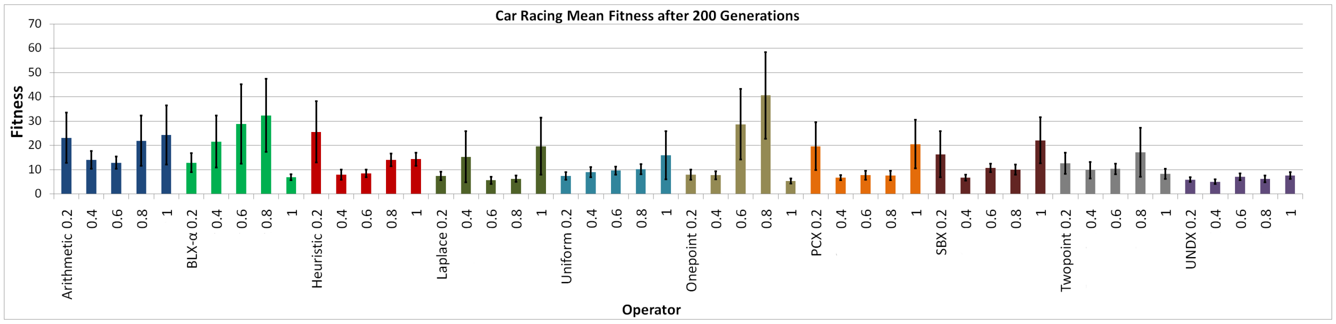

Fig. 5: Average fitness values over 30 runs obtained by the operators and crossover probabilities in the car racing simulation at 200 generations

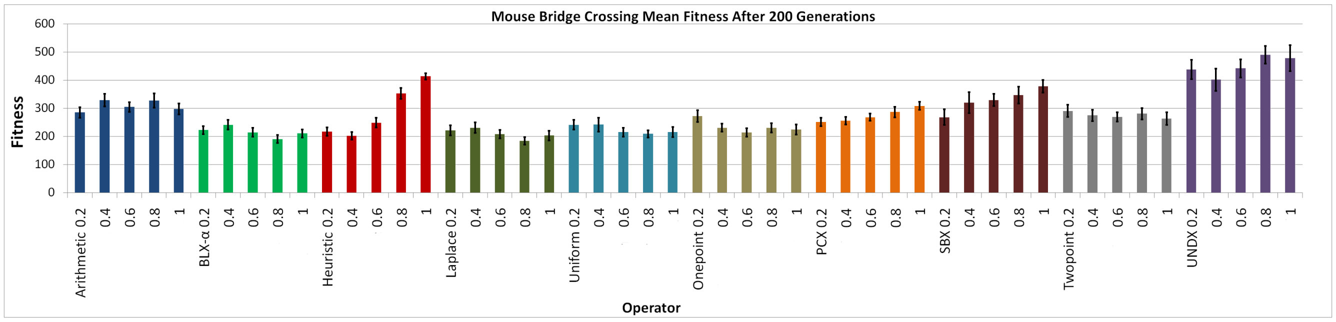

Fig. 6: Average fitness values over 30 runs obtained by the operators and crossover probabilities in the mouse bridge crossing simulation at 200 generations

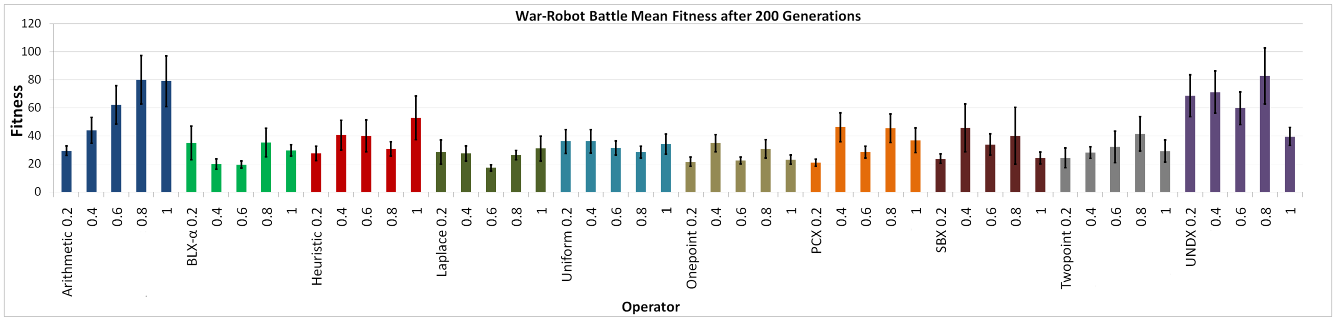

Fig. 7: Average fitness values over 30 runs obtained by the operators and crossover probabilities in the war robot battle simulation at 200 generations

genetic algorithms such as Uniform, One-point, and Two-point crossover all performed respectably across all three simulations. It would also appear that n-parent crossover operators such as Arithmetic crossover, PCX, and UNDX performed relatively poorly. As mentioned, this may be due to us using only the minimum number of parents needed for these crossover operators. However, Laplace crossover is shown to perform the best overall.

It was observed from our results that parent-centric operators performed better than center-of-mass operators, fixed parent number operators outperformed n-parent operators, and operators which combined parental information in a component-wise manner performed better than ones which use a mixture of parental information. These are possible reasons as to why Laplace crossover performs the best, however the simulation environment and task attributes are still being investigated. The results obtained are also specific to the ESP algorithm, and further investigation is required in order to generalise them to other NE algorithms.

Table I shows some of the best and worst results obtained from our tests. We interpret p-values of less than 0.05 as being statistically significant. The performance of operators can vary greatly given slight changes in the crossover probability and thus the results compared in the table should not be generalised to the rest of the crossover probabilities of that specific operator. An example of this is the Onepoint crossover in the car-racing simulation, where a crossover probability of 0.8 generates very poor performance while a crossover probability of 1 is one of the best performing operators. We refer the reader to an online appendix[7] for the complete results.

[7]https://bitbucket.org/igorawratu/neuroevolution-crowdsim

Videos of our crowd simulations can be found on our youtube channel[8].

VI. Conclusions and Future work

This study evaluated various crossover operators with a range of crossover probabilities when applied to agent controller adaptation using ESP in three different crowd simulation environments. Our research objective was to find out what operators perform well within these scenarios when used in conjunction with the ESP algorithm, and forms part of a larger goal to find what types of NE are beneficial to controlling emergent behaviours within crowd simulations. Overall we found that Laplace crossover performed the best throughout the various simulations. From our data, we infer that this is possibly due to its parent-centric, fixed parent number, and component-wise nature.

Extensions to this work include investigating the optimal number of parents for the n-parent operators, a comprehensive analysis detailing the evolutionary computation mechanisms that lead to the higher task performance of the Laplace crossover, testing the crossover operators with other NE algorithms in order to see how general the results are, studying which mutation and selection operators work well, evolving both the ANN structure and weights with an algorithm such as NEAT, investigating the feasibility of pareto-based multi-objective optimisation, and implementing a GPU version of the simulation in order to improve the fitness evaluation times.

References

[1] R. B. Agrawal, K. Deb, and R. B. Agrawal. Simulated binary crossover for continuous search space. 1994.

[2] M. Anderson, E. McDaniel, and S. Chenney. Constrained animation of flocks. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 286–297, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[3] B. Bryant and R. Miikkulainen. Neuroevolution for adaptive teams. In *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 2194–2201, Dec 2003.

[4] S. Chenney. Flow tiles. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 233–242, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

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

[6] K. Deb, D. Joshi, and A. Anand. Real-coded evolutionary algorithms with parent-centric recombination. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 61–66, May 2002.

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

[8] A. P. Engelbrecht. *Computational intelligence: an introduction*. Wiley, 2nd edition, 2007.

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

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

[11] F. J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *International Joint Conferences on Artificial Intelligence*, volume 99, pages 1356–1361, 1999.

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

[13] F. Herrera, M. Lozano, and J. L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial intelligence review*, 12(4):265–319, 1998.

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

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

[16] D. Jacka. *High-Level Control of Agent-based Crowds by means of General Constraints by means of General Constraints*. University of Cape Town, Rondebosch, Cape Town, South Africa, 2009.

[17] J. Kennedy, R. Eberhart, et al. Particle swarm optimization. In *Proceedings of the 1995 IEEE international conference on neural networks*, pages 1942–1948, Perth, Australia, 1995. IEEE service center, Piscataway.

[18] H. B. Mann, D. 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.

[19] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer, 1996.

[20] 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.

[21] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 25–34, New York, NY, USA, 1987. ACM.

[22] J. Schrum and R. Miikkulainen. Constructing complex npc behavior via multi-objective neuroevolution. *Artificial Intelligence and Interactive Digital Entertainment*, 8:108–113, 2008.

[23] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, jun 2002.

[24] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann.

[25] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. In *ACM SIGGRAPH 2006*, pages 1160–1168, New York, NY, USA, 2006. ACM.

[26] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 43–50, New York, NY, USA, 1994. ACM.

[27] D. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: An iterative search strategy for genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 77–84. Morgan Kaufmann, 1991.

[28] A. H. Wright. Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann, 1991.

[29] C. H. Yong and R. Miikkulainen. Cooperative coevolution of multi-agent systems. *University of Texas at Austin, Austin, TX*, 2001.

[8]https://www.youtube.com/channel/UCInMj1UK-1XXEo-RLidNrtw