Computational Science Technical Note **CSTN-098**

Auto-Training Animated Character Motion: A Rule-Base Tuning Hybrid Fuzzy-Genetic Algorithm

Anton P. Gerdelan

2009

The last ten years have seen animated film and computer game technology evolve to a point where controlling astronomical numbers of animated characters at once is part and parcel of a modern production. These urban crowds, armies, flotillas, and road traffic are no longer pre-calculated or hand-animated. Movement of these characters is dynamic, reactive, intelligent, and in the case of many complex systems occurs in real-time; each character is controlled by an intelligent agent. Because of its fast, efficient, and reactive properties, and its capacity for intelligence, fuzzy logic is an excellent choice of reactive motion control technology for these systems. However, there is a major drawback; each new character that a fuzzy controller is designed for has different specifications - size, speed, and acceleration - and therefore each new character requires a very large amount of manual tweaking of fuzzy set parameters, rules, thresholds, and other connected system parameters before it operates effectively (and realistically). We are interested in developing a generalised fuzzy navigation algorithm for 3D animated characters. To approach this goal we are developing a self-calibrating fuzzy controller using a hybrid fuzzy-genetic algorithm. The new algorithm that we present here is stage 2 of this target algorithm - a genetic fuzzy rule-based system that takes a new character and trains its controlling agent to operate in its intended 3D environment dynamically, in real-time, and in a parallel fashion on one CPU+GPU for very fast fuzzy rule-base tuning.

Keywords: fuzzy navigation; animated character; genetic algorithm; parallel algorithm; GPU

BiBTeX reference:

```
@TECHREPORT{CSTN-098,
        author = {Anton P. Gerdelan},
        title = {Auto-Training Animated Character Motion: A Rule-Base Tuning Hybrid
                Fuzzy-Genetic Algorithm},
        institution = {Computer Science, Massey University},
        year = {2009},
        number = {CSTN-098},
        address = {Albany, North Shore 102-904, Auckland, New Zealand},
        month = {June},
        keywords = {fuzzy navigation; animated character; genetic algorithm; parallel
                algorithm; GPU},
        owner = {kahawick},
        timestamp = {2009.12.16},
        url = {http://www.massey.ac.nz/~kahawick/cstn/098/cstn-098.pdf}
}
```

Technical Report CSTN-098

Optimising Animated Character Motion with a New Fuzzy-Genetic Hybrid

A.P. Gerdelan
IIMS, Massey University, Albany, New Zealand, and
GV2, Trinity College Dublin, Dublin, Ireland
Email: gerdelan@gmail.com

October 2009

Abstract

This paper introduces a new Genetic-Fuzzy System (GFS) that optimises animated character motion. The main innovation of our GFS is that it is able to optimise motion during run-time, which means that it is ideal for games and real-time simulations. Fuzzy controllers are currently a very popular method for animated character perception and control, and are often used to control crowds of characters in gigantic battle scenes. A major drawback of fuzzy systems is that for each new type of character time-intensive manual calibration of system parameters is required. In this paper we introduce a fuzzy-genetic system as a possible approach for addressing this knowledge gap. We also exhaustively explore the parameter space of this system to establish a set of *best practices* for use, and discuss how it can be implemented to run in the background during normal execution of a simulation.

Keywords: 3D animation, genetic algorithms, fuzzy logic, machine learning.

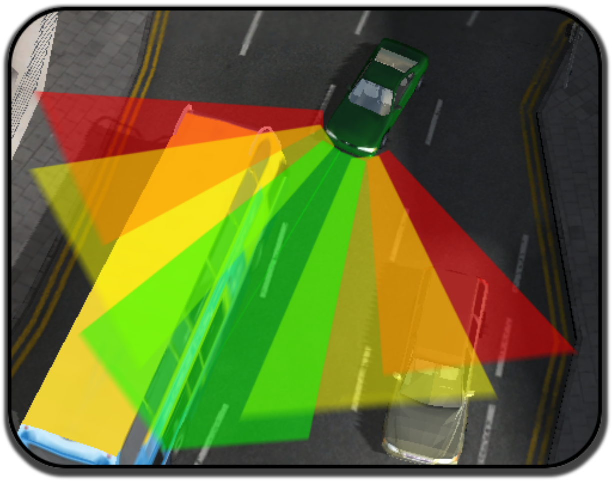

Figure 1: A car in our Dublin traffic simulation *perceives* the environment using a typical fuzzy representation (superimposed). In this case the green car is classifying the angle to other vehicles from it's heading in fuzzy terms; *narrow* green, *mid* (yellow), and *wide* (red). Objects covered by more than one set are considered a partial member of both; thus the gold car is at a **mid-wide** fuzzy angle.

1 Introduction

Fuzzy controllers are an elegant solution for intelligent motion control of real-time animated characters. Fuzzy logic produces smooth motion outputs, allows animated characters to react to changing environments and moving obstacles in real-time, and does so with minimal processing power. This technology is ideal for modern games and 3D simulations that need to simulate massive crowds of animated characters, where each character requires its own realistic motion.

Our previous works have found that fuzzy controllers for *reactive motion control* apply equally well to a diverse range of animated characters, including simulated all-terrain vehicles [1]. Fuzzy logic is the reactive AI mechanism of choice for cinematic crowd-simulation industry leaders Massive Software [2, 3], which was used for the giant battle-scenes in both the *Lord of the Rings* trilogy [4] and in the *Narnia* series of movies for both the perception and for the motion control of animated characters.

At face value it appears that a fuzzy motion system is ideal for generalisation; that we could quite easily package such a system into a *plug-in* that can control virtually any animated character or robotic application. The problem with generalised application is that each character has a unique *rôle*, physical and performance characteristics, and operating environment. This means that, while the essential kernel of the system - the fuzzy decision-making process - applies broadly, all the parameters of the fuzzy systems need to be tailored to suit each new type of animated character; be it a lumbering troll, a lane-changing city bus, or a 1940s tank surmounting a rubble-strewn battlefield. In a fuzzy system all of these variables and parameters are grouped into what is called a fuzzy **knowledge-base** (KB). The knowledge-base then breaks down into two distinct categories;

- the **data-base** (DB), which determines the size and shape of fuzzy set functions (used for fuzzification and defuzzification)
- the **rule-base** (RB), which contains a list matching every possible combination of fuzzy inputs to a valid fuzzy output

Genetic-Fuzzy System hybrids (GFS) have long been used to solve optimisation problems inherent in fuzzy systems [5] by evoling the scale and shape of either the data-base or through tuning the rule-base. However, each new GFS requires a unique **problem-dependent architecture** and **fitness function**. Fuzzy-Genetic algorithms have been used for training mobile robot obstacle-avoidance, an application domain inherently similar to simulated vehicle and animated character motion, and this approach has been shown to generate comprehensible and reliable fuzzy rules through this method [6]. Although some works have begun to explore GFS for autonomous agent motion in very basic stochastic applications [7], using a GFS for automatic training of 3D animated character motion is a new problem domain, and as such we have a designed a new GFS framework.

2 Method: Fuzzy Controllers in Animation

The primary use of fuzzy controllers is to simplify an agent's understanding of its environment. Instead of classifying distances and angles in terms of meters and degrees, for example, we classify angles in human-like terms as being members of fuzzy sets *narrow, mid* or *wide* and distances *near, medium* or *far*. This classification is called *fuzzification* and is done by taking the real input values and evaluating their fuzzy equivalents using fuzzy set membership functions.

An example of this procedure is illustrated in Figure 2. Once we have simplified perceptions into discrete forms like this, then we can perform some human-like reasoning by matching our inputs together. An example fuzzy rule:

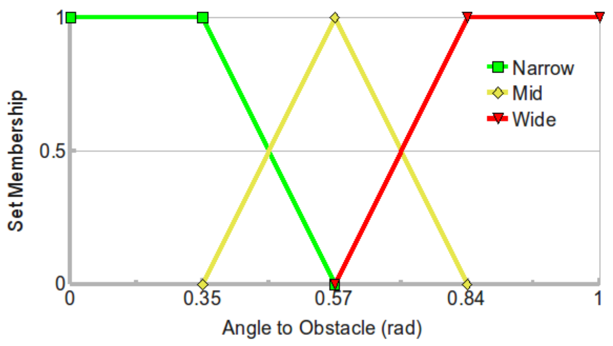

Figure 2: This example of fuzzy input set functions is from the city bus in our traffic simulation. It takes a real angle to an obstacle in radians, and fuzzifies this into either full or partial membership in the *narrow, mid* and *wide* fuzzy values. For a 3D-spacial representation of this classification see Figure 1.

*"if a car is **near** and the angle to it is **narrow** then steer **sharply** away"*

Most intelligent agent systems would use a mathematical function to match inputs to outputs, but with a fuzzy system we can use a table to look up our rules very quickly. This rule table, which is known as a *Fuzzy Associative Memory Matrix*, or FAMM, matches each fuzzy distance and angle to fuzzy output values. As an example, the FAMM that we are using for change to steering in the *route-following component* of our simulated bus is given in Table 1.

	narrow	mid	wide
near	sharp	medium	very light
medium	medium	very light	zero
far	very light	zero	zero

Table 1: FAMM for change to steering in the *obstacle-avoidance component* of a simulated Dublin bus. Output fuzzy steering adjustments are given for each fuzzy input distances and angle combination.

To cover cases where inputs are a partial member of multiple input sets because the input values fall inside overlapping set membership functions (such as the gold car in Figure 1) we evaluate *all* of the rules, and *aggregate* the outputs together using a centre of mass function weighted by the degree of membership in each fuzzy input set, and is of the form:

$$output_{crisp} = \frac{m_0 * w_0 + m_1 * w_1 + ... + m_n * w_n}{w_0 + w_1 + ... + w_n} \quad (1)$$

Where m is a *mid-value* (also known as a *singleton value*) for a fuzzy output set; for example *sharp* turn might have a mid-value of $2rad \cdot s^{-1}$, and *very light* turn might have a mid-value of $0.5rad \cdot s^{-1}$, and where w is the weight

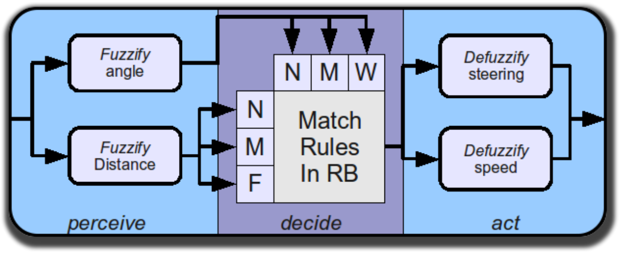

Figure 3: Our **fuzzy decision-making component** architecture, which has two fuzzy perception inputs (*angle* and *distance*), a 3∗3 rule table which matches input fuzzy sets to output fuzzy sets, and two fuzzy motion defuzzifiers (*steering adjustment* and *desired speed*) which aggregate the values of fuzzy output sets together and produce a crisp output. Larger systems stitch thousands of decision-making components together into a massive graph, but we are using only two such nodes in our system.

value, between 0 and 1, of a particular fuzzy output set. This output procedure is called *defuzzification* and produces a final real (crisp) value. In our car steering example this crisp output will be a steering adjustment value in radians.

The basic fuzzy decision-making architecture that we are using for all of our animated characters is illustrated in Figure 3. Complex fuzzy logic *brains* for movie characters contain trees of thousands of cascading fuzzy decision-making nodes of this type [4], but our systems are considerably simpler case and comprise only two of these fuzzy decision-making nodes:

- A reactive **obstacle avoidance** controller
- A **target-seeking** or route-following controller

Using both of our decision-making modules, environment elements (obstacles and destinations) are fuzzified into input values for angle and distance. Once these have been obtained we match the fuzzy distances near (N), medium (M), and far (F) to the fuzzy angles narrow (N), mid (M), and wide (W) in our FAMM. We have found that 3 sets for each input is sufficient (which gives us a $3x3$ rule table), and that the more larger, more detailed rule-bases are superfluous in this kind of application. Our rule-base (RB) contains the complete matching for every input, and provides an output fuzzy set for each rule, for both *change in steering*, and for *desired speed*. Therefore, with 2 fuzzy decision-making modules that have $3 * 3$ fuzzy inputs apiece, we have 18 combinations of inputs, and as each of these combinations is used for 2 fuzzy outputs then in total our fuzzy system requires 36 fuzzy rules.

3 Genetic-Fuzzy System Design

Besides our modest aim of automatically tuning the motion control for an animated character, evolutionary algorithms have the potential to tackle several *"Holy Grail"* problems for computer animation and games, and we have designed our system to investigate these as well:

(1) One size fits all - Given that our fuzzy controller model applies to a diverse range of animated characters, should the GFS significantly improve the motion of one character, then it has the potential to become a *generalised intelligent motion control system for animated characters*, that can automatically calibrate itself to suit a given character's peculiarities, environment, and rôle. A human designer would then only need to adjust the parameters of the fitness function to produce desired behaviours, rather than tinker with the vast array of fuzzy parameters. In the case of our target city traffic simulation we have a large number of vehicles of different shapes and sizes - from sedans to buses to giant amphibious tour vehicles - and it would be very useful if we could provide one rough set of movement rules and have it fit itself to the movement requirements of each type of vehicle.

(2) Run-time adaptive learning - The belief has long been held that genetic algorithms are far too resource-intensive to learn *during run-time* of a simulation or game [8], and that a very large number of generations and very large populations are required. Optimisation of artificially intelligent characters is usually performed in long, objectively designed and evaluated staged batches prior to actual use. The major notable exception is the NERO game [9, 10], which uses a human-guided training system and operates in real-time. There are limitations to the existing pre-trained paradigm; it can not adapt to suit new conditions in real-time, which means that characters are unable to cope with any change to operating environment or to a new scenario after the initial training phase is finished. This means that if a very complex environment is developed in a film scene, for example, and the characters are trained in a long process to operate optimally in this specific environment, then any later changes to the scene (which in practice will always occur) would require recalibrating the brains of all the agents. It would be much handier if the characters were able to simply adapt themselves to the environment as design decisions changed.

(3) Interacting with human players - If we can show that a GFS can operate on commodity hardware in tandem with a fully graphical game-like 3D simulation, and that it can make some improvement to fitness in a small number of hours, then we can replace the idea of a *pre-packaged AI* paradigm with one where machine learning in entertainment can be *human-interactive*. This has implications for more complex systems such as tactical computer game opponents. If the fuzzy systems are simple enough to evolve within game time constraints then a new set of tools can be developed to challenge and immerse humans in a more interactive, and personalised virtual world.

3.1 Finding the Threshold of Shortest Evaluation

Before we can attempt run-time adaptive learning we need to establish how long it takes to evaluate each individual. As a control study we set up a simple experiment to mimic typical 3D graphical computer game conditions. A number of characters were moved through an obstacle course, taking approximately 2 minutes of simulation time per run through the course. The characters were all given pseudo-random starting positions and orientations at the start of every run and had to cope with varied terrain and obstacles. We created a simple self-evaluation heuristic *score* that penalised characters for intersecting with obstacles, and awarded points for shorter times to the destination.

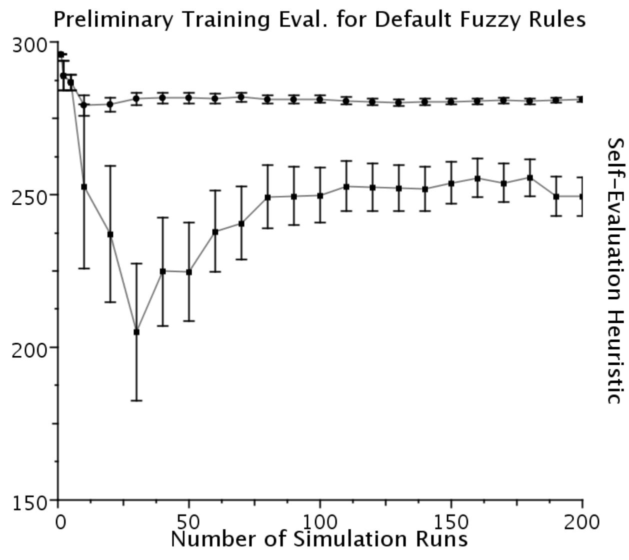

Figure 4: *How many runs is enough to establish a reliable fitness estimate?* The first (lower) plot was unreliable, and didn't stablise until around the 70 run mark as a number of technical difficulties were producing -1 (failed) evaluation scores. After removing these, and re-plotting the graph, we can see that we already have a useful score heuristic by only **20-30 runs**.

No evolution took place, and all agents were using the same fuzzy KB. We were interested in how many simulation runs of this length needed to be accumulated before a reliable estimation of fitness was found. Figure 4 presents the results of this study. An important side-effect of this study was that it highlighted a flaw in our evaluation method - we were awarding a score to individuals *on completion* of the run. Individuals that did not arrive at the destination location were awarded a -1 (complete failure) score. Observation of the process showed that, whilst overall motion performance of the characters was good, there was a small flaw in the rule-base that resulted in a large number of characters stopping *just short* of their destination. The lower plot in the figure gives us these results - which contain an enormous amount of error, due to the disparity between -1 scores and those in the $280 - 300$ range. As such the score heuristic did not stabilise until after 100 runs. Because we are actually interested in fostering quality of motion-in-transit of the characters, and not whether they arrived at the exact target destination, we compensated for this oversight and replotted the graph. Nevertheless we discovered an important principle in the design of fitness functions for dynamic 3D environments - *rewarding agents with pass/fail criteria is not sufficient - you must evaluate based on continuous motion.* We can see in the amended (upper) plot line that a reliable fitness evaluation was then obtained in only $25 - 30$ runs of this type.

3.2 Evaluation Method

"It is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change." - Charles Darwin [11]

Our experiments have shown that the current state-of-the-art approaches for optimising animated character motion are highly ineffectual, and fraught with technical difficulties. Our initial approach was to use batch-driven optimisations, where the rule-base of the agent was tweaked ±1 level of fuzzy output value in continuous simulation, and in which characters were put through a prepared *obstacle course* representing their target operating environment. Upon completion of the course, characters would be restarted at a new starting position and orientation. We identified the following problems with this method:

- Special obstacle courses need to be prepared for training
- Training is hard to distribute across multiple machines
- Lots of infrastructural changes are required to the target simulation or a separate training simulation is required.
- This kind of training could never tailor itself to particular human controller.
- The system is unable to adapt to later change in the environment

It has been proposed to comparatively benchmark the performance of motion algorithms across a range of abstracted case-scenarios [12, 13]. This is a promising approach for evaluating our own motion system, and using this evaluation as a fitness heuristic to drive our optimisations. We could then simply look up in score a table to see which algorithm best suits our environment, or as the designers suggest with their formula, find an overall best-scoring steering algorithm over all tests. However, because animated characters have such a diverse range of applications and motion types we propose that it a *fitness for purpose*, or a *run-time evaluated* approach to application that is taken. Because we intend to evaluate the motion of our characters continuously, and in real-time, this means that our evaluation heuristic is completely subjective and can not be compared between different systems, and that even within one simulation, if the environment changes over time then so will the evaluation. This approach does have the unique advantage that a character can *adapt* to suit new conditions, or to a particular human player in game.

3.3 Architecture of the Dynamic GFS

The KB of a fuzzy system is not a homogeneous structure but is rather the union of qualitatively different components; the rule-base and the data-base. Thus Genetic Fuzzy Systems are designed to operate on either part [5]. In this architecture we have chosen to only concentrate on optimising the rule-base, as output fuzzy values can be treated as discrete numbers, and are therefore easy to increment or decrement during mutation. We consider data-base optimisation to be the more complex challenge, and will thus investigate scaling and modification to fuzzy set functions in future works.

Our first therefor, is to create a genetic structure representation (chromosome) for our fuzzy rules. Our genetic operators (*selection, crossover,* and *mutation*) can then treat our rules as if they were biological processes dealing with genetic code. In our encoding method each rule is recorded as a three-digit cluster. The first digit represents a fuzzy input value for *fuzzy input angle*, the second for *fuzzy input distance*, and the third is a *fuzzy output value*. Both input and output fuzzy values are expressed as integers rather than a fuzzy name; e.g. *zero* is expressed as 0, and a *very light* turn as a 1. This makes manipulation very easy, as the entire rule-set of 36 rules is simply a string of characters, whilst it retains some human readability. To aid in identifying individuals later we have added a genetic *ear tag* to the front of the code, which indicates which generation and batch of runs the individual came from. An example individual's chromosome from our experiments is given in Algorithm 1, with rules separated by spaces for clarity.

We have designed our in-simulation architecture with a view to making minimal intrusions on the target simulation for two reasons. Firstly, we identified in Section 3.2 that one drawback of existing genetic algorithm models is that they require extensive changes to simulation infrastructures. We suggest a light plug-in approach, which integrates seamlessly into the existing fuzzy architectures, requires very little CPU overhead, and has only a very small dependency on the target simulation's programming language and implementation. Our simulation plug-in module is illustrated in Figure 5.

Algorithm 1 The chromosome for an individual of the DUKW all-terrain mode species. This individual is batch 0 (B:0) of generation 1 (G:1). Each 3-digit cluster represents a fuzzy inference rule. Each set of 9 rules is a complete mapping for a fuzzy output. Our genetic algorithm operates by changing the value of the third digit (the rule output) of the 3-digit clusters using our genetic algorithm.

```
G:1 B:0
021 122 223 011 113 214 001 014 025
225 124 023 214 113 012 201 102 001
000 011 022 100 112 123 200 213 224
005 014 022 104 113 121 202 211 220
```

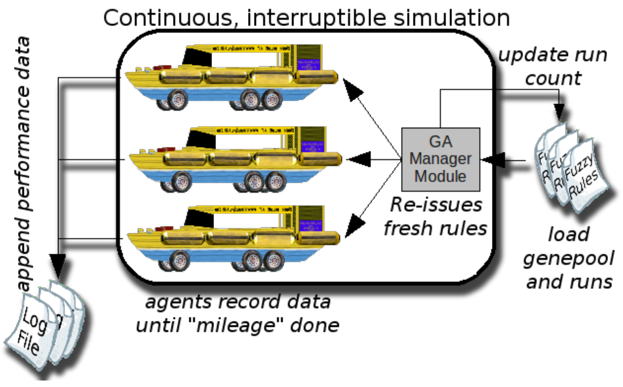

Figure 5: Dynamic evolution during real-time simulation execution. Real-time evolution is feasible as GA demands of the simulation itself are extremely lean; actual *evolutionary* functions are managed in an external process (see Figure 6). Here a light-weight module loads new fuzzy rule sets and distributes these to character-controlling agents on the fly. The agents output their performance data to logs at regular intervals.

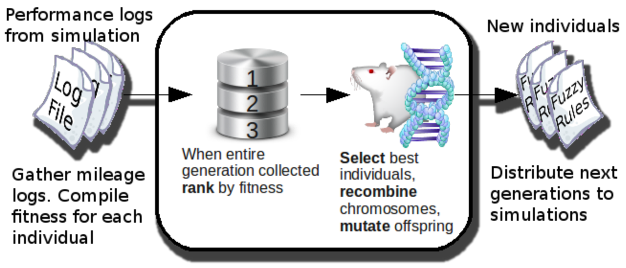
Performance logs from simulation
Log File
Gather mileage logs. Compile fitness for each individual
1
2
3
When entire generation collected rank by fitness
Select best individuals, recombine chromosomes, mutate offspring
New individuals
Fuzzy Rules
Distribute next generations to simulations

Figure 6: Operation of the breeding tool chain. This pipeline runs independently of, and asynchronously to the subject simulation. Firstly, log files from simulations are read and compiled into *training runs*. A fitness score is computed for every training run. When all runs are completed every member in the population is ranked by fitness score against those from older generations. Top-ranking individuals are **selected** for reproduction, and their offspring **mutated** and distributed to simulations.

A small module loads all available *chromosomes* into the simulation on programme execution. During run-time it distributes these to the agents, and once all runs are completed it looks for a new generation of rule-sets to load and distribute on-the-fly. Agents log performance (fitness function components) at regular *mileage* intervals; these are treated as small parts of training runs. These can then be compiled at a later stage for fitness evaluation. This implies continuous evaluation that is both subjective, and also *interruptible*, which makes it ideal for computer games where a game might not last for an entire evaluation cycle. Because the runs are split into small segments and logged incrementally the training can then be resumed at a later stage with very little training time loss.

The actual *breeding* of new generations based on genetic operators is handled in an external pipeline, as illustrated in Figure 6. This pipeline is then not dependent on the implementation of the target simulation and can operate quite independently of simulation. Once independent of the simulation the CPU demands of the whole system are very low, as the breeding pipeline does not need to remain in-synch with the updates of the simulation. It can also readily take advantage of any available multi-core hardware by occupying a separate process altogether.

This separate tool-chain reads in all of the logged evaluation results output by the simulation and compiles them into complete runs. The runs are then evaluated with a *fitness function*, and when all of the runs for an individual are compiled, then it is ranked according to its fitness score. The fact gene-pool in our architecture *retains the best individuals* from earlier generations, and new individuals are ranked against these.

As for **genetic operators**, our *selection* operator takes the best p parent individuals from the top of the fitness rank and these are used for breeding the new generation. For the sake of broad applicability (so that the algorithm functions consistently even when we experiment with very small population sizes) we have only used a value of $p = 2$ in our experiments so far. The parent chromosomes are then *crossed over* to produce a population size of n children. The cross-over mechanism moves along the chromosome one rule at a time and has a set 50% chance of choosing either parent's output fuzzy set for each rule. Each rule has an r (*radiation-level*) chance of it being *mutated* by 0 to m fuzzy output levels. We have attempted to exhaustively explore the complete range of genetic algorithm variables, the results of which are presented in Section 4.

The **fitness function** that we have used for our initial experiments is has been designed to be as simple as is practical, and is given in Equation 2 as the fitness awarded to an individual i.

$$fitness_i = \bar{c}^2 * w_c + (1 - \frac{\bar{v}}{v_{max}}) * w_v \qquad (2)$$

Where $\bar{c}$ represents our "crash-rating" - the mean penetration or intersection of the character with obstacles during the run in meters. We leave this in its original squared form (distance comparisons in most simulations are usually squared to avoid use of the CPU-expensive square root operator) for expediency. And where v represents the speed of the vehicle. In our case we are taking the mean speed of the vehicle over the maximum desired speed - the "ideal" speed v_{max}. Each of the equation components are multiplied by a weighting factor (represented by w_c and w_v) which can be used to reward obstacle-avoidance behaviour or expediency to a higher or lower degree. For our initial experiments we have left these weights set to 1. Our perfect fitness is a score of 0 so we aim to minimise the fitness. For an example fitness evaluation, if we have a crash rating of $0.4m^2$ and an average speed of 20 out of an ideal $30k \cdot h^{-1}$ then we award the individual a fitness of 0.7333.

4 Experiments and Results

We designed a range of experiments in-simulation to attempt to disprove our hypothesis *"That a dynamic genetic-fuzzy system is a feasible approach for optimising a fuzzy rule-base"*, and also to exhaustively explore the larger part of the parameter space of the genetic algorithm in order to establish a range of *best practices* for employment of the dynamic GFS.

All of our experiments were conducted in a free-roaming 3D graphical simulation that we constructed using the OGRE3D [14] graphics library, where we allowed simulated to-scale 1940s tanks to pseudo-randomly traverse an obstacle-strewn environment similar to that found in many modern computer games. The vehicles were accurately simulated with historical performance data, and a simple

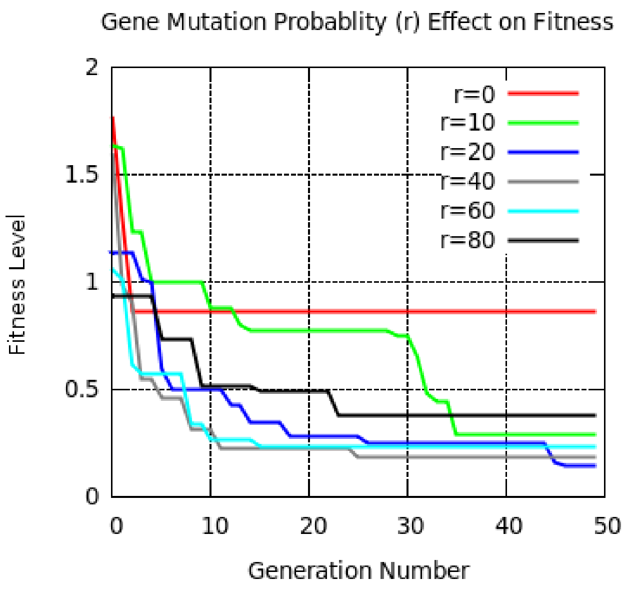

Figure 7: We can see that our mutation probabilities in the range $20-60\%$ are the fastest learners with a *"sweet spot"* at $r = 20$ breaking the minima trap by 45 generations.

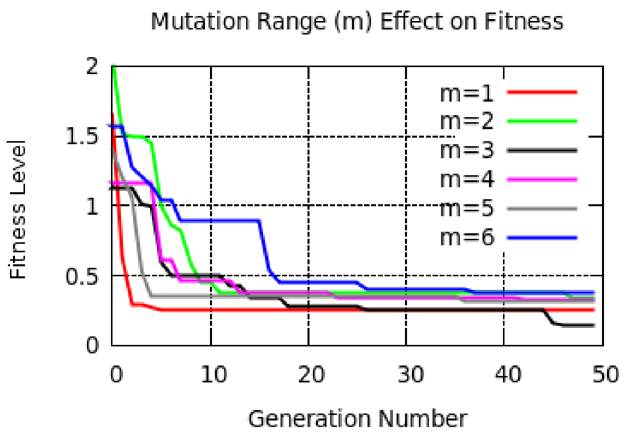

Figure 8: We discovered that the level of gene mutation made very little difference to fitness improvement, except that high-levels $(5-6)$ produced a lot of useless individuals that were unsuitable for run-time use.

physics simulation applied to braking, acceleration, turning friction, hill climbing and descent. Thus we put our fuzzy-controlled agents in a typically complex game or film-type environment that they had to perceive and operate in using a very simple fuzzy system. We found that we were able to quite comfortably distribute our training over 40 characters simultaneously on a commodity desktop machine without adversely affecting the frame-rate of the simulation or overly cluttering our test environment.

At the start of each experiment our characters were randomly scattered around the landscape, and continuously given destinations to move to. The characters had to avoid a large range of shapes and sizes of static and dynamic obstacle, including long walls, and other moving vehicles, whilst being forced to move through steep craters, hills, and flat areas. The simulation was not restarted between evaluation runs, but rather the new runs were awarded randomly to available characters in an uninterrupted fashion. In all of the experiments we started all of the agents with the same default hand-crafted fuzzy rule-base, which was capable of motion but was able to be greatly improved. We designed a series of experiments which would evaluate our genetic algorithm's variables; *mutation range (m)* where each fuzzy output value to be mutated would be adjusted $\pm m$ levels, *probability of gene mutation (r)*, and *population size (n)*. We based all our experiments on our preliminary study (see Figure fig:prelim-cleaned) which showed that 20-30 runs was sufficient for minimising error in our evaluation, thus we repeat the evaluation of each generation 30 times in every experiment.

Our first experiment was designed to find the ideal probability of gene mutation *r*. All of the other genetic algorithm variables were kept constant; with $m = 3$, and $n = 4$. As we were evolving during a continuous simulation we included a control case $r = 0$ to observe how the changing condition of the simulation was affecting the fitness evaluation itself. The results are presented in Figure 7. We ran our experiment with 6 different "radiation" levels. The control case showed us that our simulation stabilised after 3 generations time as the characters tended to spread themselves further apart. At this point we have a base fitness level of 0.86. The extreme cases; $r = 10$ and $r = 80$ were slowest learners, with all results tending to a local minima trap around a fitness of 0.3. Only $r = 20$ broke through this trap by 45 generations. Overall the results indicate that the genetic algorithm was making a significant improvement to fitness over the control case. The difference in rate of improvement to fitness between over the entire range of probabilities of mutation is marginal, with a sweet spot around $r = 20$.

Our second experiment was designed to explore the parameter space of the mutation level *m*. We hypothesised that our m variable would be the most important in our genetic algorithm, and that higher the mutation levels would punch through minima traps discovered at lower levels. Our experiment conditions were the same as in our previous experiment, but with r held constant at 20%. The results of this are presented in Figure 8 where we can see that the data is *not consistent* with our hypothesis. In fact, lower mutation levels of 1-2 were adequate for quickly tuning the rule-base. All of the different levels were trapped in our familiar 0.3 minima, with only our $m = 3$ breaking through this within 50 generations. An interesting observations of mutation levels was that where the mutation level was set to higher levels of 5 or 6 a huge number of individuals produced were completely useless and remained stationary or quivering with malformed steering brains. As mentioned

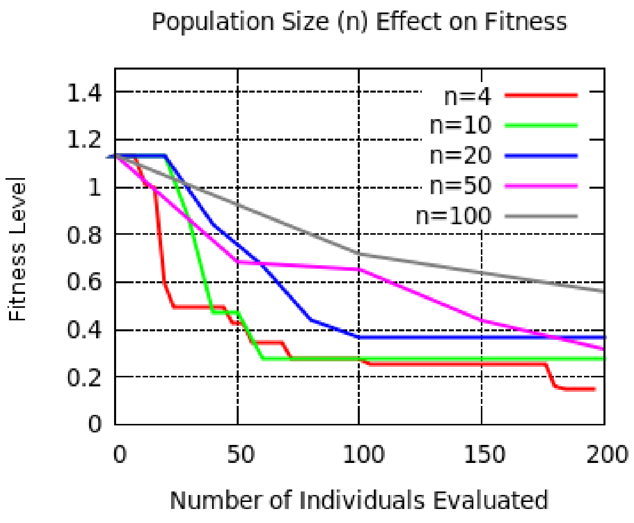

Figure 9: Our population size experiment was designed to find the best rate of ftiness improvement (note that the x-axis measures the number of individuals evaluated, not the generation number).

earlier we had to develop a special escape clause to eliminate these individuals in order to continue the experiment. Although $m = 6$ was capable of making huge jumps in fitness (generation 15 for example), it can also produces stretches with no improvement and many useless characters (it took 15 generations before any significant improvement was made to fitness). This behaviour would be highly undesirable in an interactive computer game, so we can say that, whilst high mutation levels may allow pre-trained rule-bases to punch through minima traps over a very large number of generations, for run-time training where consistently *good* motion is desired a *rule-base tuning* approach is better, with mutation level set in our case between $1 - 3$.

The third experiment was designed to explore the population size (n) parameter space, and our aim was to find the population size per-generation that allow us to improve our fitness score most rapidly - *by evaluating the smallest number of agents*. We hypothesised that larger population sizes would improve our rate of fitness improvement beyond our very low default of $n = 4$. Our results are presented in Figure 9, we can see a clear indication that *bigger population sizes evolve more slowly*, directly contradicting our hypothesis.

5 Conclusions and Future Works

In this paper we have shown that our GFS architecture can be used to effectively optimise the motion of 3D animated characters. We have also found that it can also operate dynamically, in a real-time simulation, and evolve with a small population size. This offers an alternative to the commonly held assumption that machine learning is unsuitable for run-time optimisation in 3D graphical simulations and games as it too CPU-intensive, takes too long, and requires huge populations.

In addition to presenting our complete GFS architecture, we have also identified a range of *best practices* for use, and found the ideal range for genetic algorithm parameters in this type of application - $m = 3$, $n = 4$, $r = 20$.

Our success at training our experiments' characters indicates that our architecture is a strong candidate for a *generic* auto-calibrating controller for animated character motion, and we intend to investigate the full breadth of possible application in future works including flocking animal simulations, pedestrian crowd simulations, and traffic simulations.

Acknowledgements

Thanks to the GV2 group at Computer Science, Trinity College Dublin, Ireland with Science Foundation Ireland for hosting and supporting this work, and to the Institute of Information and Mathematical Science at Massey University, New Zealand.

References

[1] A. P. Gerdelan and N. H. Reyes. Towards a generalised hybrid path-planning and motion control system with auto-calibration for animated characters in 3d environments. *Advances in Neuro-Information Processing, Springer Verlag Lecture Notes in Computer Science*, 5507:25–28, November 2008.

[2] Jordi Bares. Massive software 2.0: Intelligent agents aid animators. *Reel-Exchange*, 1:1, Jul 1 2005.

[3] Massive Software. Feature list. http://www.massivesoftware.com/feature-list/, October 2009.

[4] New Line Productions, Inc. Massive brains. inside the effects: The battles of middle earth. multi-media website, 2002.

[5] O. Córdon, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena. Ten years of genetic fuzzy systems: current framework and new trends. *Fuzzy Sets and Systems, Elsevier*, 141(1):5–31, 2004.

[6] M. Mohammadian and R. J. Stonier. Fuzzy logic and genetic algorithms for intelligent control and obstacle avoidance. *Complexity International*, 2:ch 2, 2005.

[7] Rogério Neves and Marcio Lobo Netto. Evolutionary search for optimization of fuzzy logic controllers. In *1st International Conference on Fuzzy Systems and*

Knowledge Discovery, volume 1 of *on Hybrid Systems and Applications I*. Springer, 2002.

[8] Greg James. Using genetic algorithms for game ai, October 2005.

[9] Risto Miikkulainen. Creating intelligent agents in games. *The Bridge*, 36(4):5–13, 2006.

[10] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, December 2005. Department of Computer Sciences, The University of Texas at Austin.

[11] Charles Darwin. *On the Origin of Species*. John Murray, Albemarle Street, London, 1859.

[12] Shawn Singh, Mishali Naik, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinmann. Watch Out! A Framework for Evaluating Steering Behaviors. *Lecture Notes in Computer Science: Motion in Games*, 5277/2008:200–209, 2008.

[13] Shawn Singh, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinmann. SteerBench: a benchmark suite for evaluating steering behaviors. *Computer Animation and Virtual Worlds*, 1:1546–4261, 2009.

[14] G. Junker. *Pro OGRE 3D Programming*. APress, 2006. ISBN 1590597109.