OPTIMISING OFFENSIVE MOVES IN TORIBASH USING A GENETIC ALGORITHM

Jonathan Byrne, Michael O'Neill, Anthony Brabazon

University College Dublin
Natural Computing and Research Applications Group
Complex and Adaptive Systems Lab
University College Dublin
jonathanbyrn@gmail.com,m.oneill@ucd.ie,anthony.brabazon@ucd.ie

Abstract: *Game AI is a collection of techniques that imitate human intelligence in non-player characters (NPC). The conventional approach to game AI is to implement a specific intelligence for a specific game environment. In this paper we evaluate an adaptive approach that creates components for game AI. Evolutionary algorithms (EA) have shown their capability for developing successful strategies on a wide variety of problems. This paper details the application of a genetic algorithm (GA) to evolve opening moves for the turn-based fighting game, Toribash. The aim of this paper is to show the applicability of an evolutionary algorithm for game AI and how its adaptive nature allows it to be applied to a wide variety of game environments. This work also highlights the suitability of Toribash as a testbed for further EA research.*

Keywords: *Genetic Algorithms, Artificial Intelligence, Games*

1 Introduction

Developing AI for games is normally a bespoke affair. Instead of the striving for a generalised AI like other fields of machine learning, the settings in which it has to operate i.e., the game environment, is highly restricted. This means most implementations only exhibit specific intelligence for a particular game. Although this means that the problems facing the machine learning algorithms is dramatically reduced, it also results in brittle game behaviour; any change to the game environment can result in the game AI functioning poorly. One solution to this is to use a machine learning approach to create the AI for the game. Our aim is to implement a GA capable of producing realistic AI behaviour. This adaptive approach means that changes to the game environment can be accommodated by allowing the algorithm to adjust the AI, thus minimising human intervention. The direct benefits from this approach are increased game playability and a reduction in development time. This is an area of growing interest as users expect to see both realistic and unpredictable behaviours in game agents.

One area that has only seen limited AI development is fighting games. Fighting games involve a one-on-one engagement with an opponent in real time. The game environment is highly confined, it is normally a small arena that allows only 2 dimensional movement, and the selection of moves is limited to a small set for each player.The result is that the computer characters only require limited AI, such as a finite state machine or a decision tree, to match human performance [10]. Toribash is different. It is a turn based fighting game that allows you to create your own moves instead of selecting predefined moves. There is also an active community that create new game environments that are known as mods. The open nature of Toribash and continuous additions of new game modes makes it a difficult challenge for any single machine learning technique to produce consistently good results.

Evolutionary algorithms use an iterative process that incorporates variation, selection and inheritance. The algorithm creates a 'population' of possible solutions, selecting the best individuals to create the next generation. The process is driven by a fitness function, which is used to evaluate the solutions. The fitness function only specifies how to evaluate the output, not how to accomplish it. By allowing EA's to explore the problem space, they are capable of creating novel solutions. EA's are suited to open ended problems like Toribash as they are capable of creating innovative solutions with no advanced knowledge of the game environment. Our intention is to apply a genetic algorithm to Toribash and observe whether it is capable of evolving high scoring moves in different game environments.

This paper is organised as follows, Section 2 summarises previous applications of adaptive learning techniques. Section 3 describes the game play of Toribash and reasons for investigating it. Section 3.1 discusses our experimental settings, our choice of fitness function and why we think a genetic algorithm is suitable for this problem. Section 4.3 and 4.4 describe the experiments carried out on Toribash. Finally, in Section 5 and 6 we discuss our conclusions and future work.

2 Related Research

Game AI has existed since the first generation of video games. The earliest examples of game AI were arcade games such as Pac-Man, Space Invaders, Donkey Kong and Joust. These used basic rules and scripted sequences combined with randomisation, to generate simplistic behaviours. From these modest beginnings a rich and varied level of different AI types have developed but it is only recently that adaptive learning techniques have been used. One of the first commercially successful applications was the game Creatures by Stephen Grand [7]. Creatures used neural networks to govern the behaviour of the in-game characters called Norns. During the course of the game the Norns would procreate and their offspring would inherit their parents behaviour.Black and White [9] also used Neural Networks for the AI of their in-game agents. Black and White is a real-time strategy game that features a unique element, an agent that you can raise and train to do you bidding. This was a very successful example of machine learning algorithms operating during the execution of the game. Another example of a combination of Neural Network and EA is also seen in Galactic Arms Race [8]. Galactic Arms Race uses the cgNEAT algorithm to create game contents such as weaponry based on the players preferences. Currently game AI tends to focus on tried and tested methods such as Finite State Machines and decision trees [10] because of their speed but there has been a move towards more stochastic techniques to elicit more realistic behaviours from opponents. Evolutionary Algorithms have been applied to every level of agent behaviour [4] [10]. GA's are used to weight an individual agent's actions in first person shooters [5] and Genetic Programming (GP) has been used to evolve high level squad coordination behaviours between agents [6]. The customer demand for realistic yet unpredictable computer AI means that this is an area of growing interest.

3 Toribash

Toribash [2] is a turn based fighting game that was originally created by Hampus Söderström. It differs from traditional fighting games as it does not have moves specified in advance for the user. Instead, the game uses a rag-doll physics engine that allow the user to specify the state of the rag-doll's joints. In single player mode the user's name is Tori and the opponent is called Uke. Each joint can be either extended, contracted, relaxed or held. The user changes the state of the rag-doll's joints after a specified number of time steps and the match finishes after the total number of time steps has been reached. The winner is the player who inflicted the most damage on their opponent while avoiding being disqualified. Disqualification occurs when a player comes in contact with the ground with any limb other than their hands or feet, or when the player leaves the designated arena.

The game has been freeware on both PC and Macintosh since 2007. The game is capable of running additional game environments called mods. It has a large and active modding community and an open framework for creating both mods and scripts for player behaviour in the Lua programming language. Different game settings can be specified by the mods, such as altering the number of times steps in a round, the size of the arena, and the initial starting distance between players. The mods can also completely change the game mode. There are mods that alter the player size and shape, add weapons, change the games physics and add objects to the arena. Some examples of this are shown in Figure 1. The game currently comes with over 1100 mods implementing various game styles such as; free running, acrobatics, skating, jousting, karate, frictionless environments and sword fighting.

Access to the game events and settings is through the game API, which is available on the website. The API allows you to get information on the game state, the joint states and character information. There are also hooks that allow you to interrupt game play and execute your script after different game events. Toribash poses an interesting problem for any machine learning algorithm. There is an upper bound on the possible player configurations but the game play itself is open ended. There are feedback mechanisms such as player position and player scores and there is an abundance of game environments, each with a different fitness landscape. For these reasons we find Toribash a flexible testbed for our experiments.

3.1 Application of a Genetic Algorithm to Toribash

At its most basic level, Toribash is a combinatorial problem. There are 20 joints with 4 possible arrangements and 2 joints with 2 possible arrangements, as shown in Figure 2. For a single move there are 4.4 trillion possible combinations. To investigate every move would take an unreasonable amount of time. A better approach is to use a genetic algorithm. A GA samples the search space and then explores the areas that produce the best results. While it may not find the global optimal solution it is often capable of finding a satisfactory solution. It is this ability to hone in on effective solutions that make it suitable for Toribash. There are such a plethora of game types that no single approach could work for all of them. Training bots have been hand coded for certain game types but they lack generality and even small changes to the game rules can cause them to malfunction.

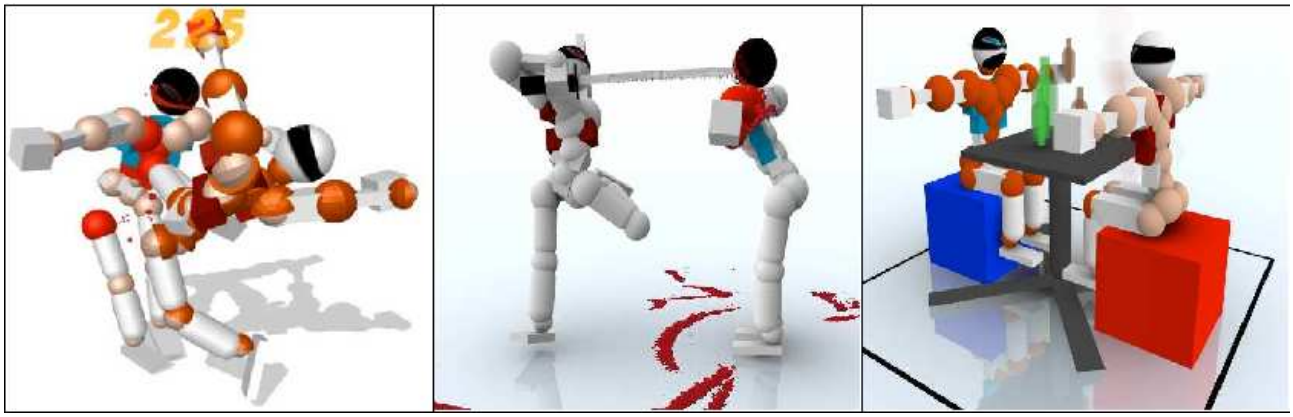

Figure 1: Different game environments: standard, sword fighting, and bar room brawl

An evolutionary approach avoids this problem by making no assumptions about the game framework. The EA learns each game type through trial and error which should allow it to adapt to any game setting. This is what we hope to show in our experiments.

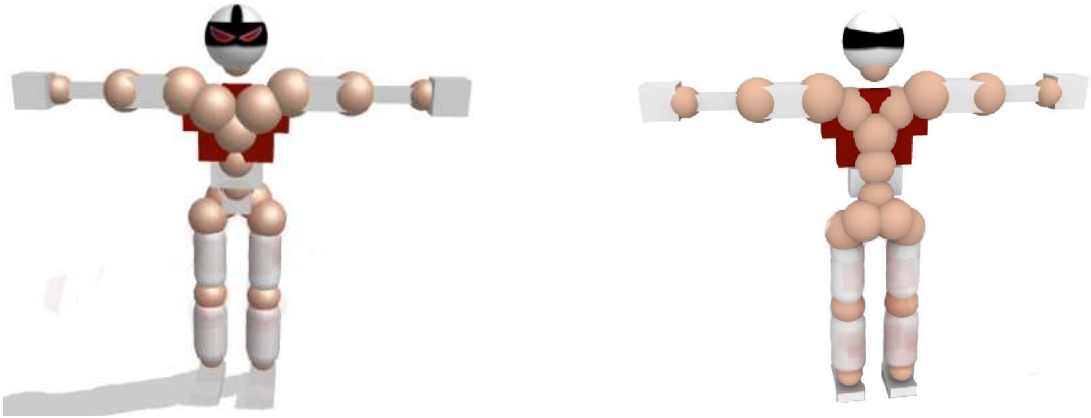

Figure 2: Tori's joint positions

4 Experiments and Results

Our intention in applying a GA to Toribash is to examine whether it is capable of obtaining a high score in a sample of different game environments. A secondary goal is to explore how the GA traverses the search space and if it is capable of developing rich and varied AI behaviour. In this section we describe how we set up our experiments, the experiments that were carried out and the results we obtained.

4.1 Experimental Setup

The experiment was carried out by a GA implemented in Lua. The source code for the GA may be downloaded from the forum [3]. The Toribash scripting interface executed the GA to generate moves. Each move consisted of a chromosome with 22 integer codons. Each codon specified a particular joints current state, whether it was extended, contracted, held or relaxed. The joint positions were not grouped in any particular order and instead used the configuration that was originally specified by the Toribash API as shown in Table 1. In all our experiments the behaviour of the Uke bot is kept static. The GA evolved a population of size 50 over 50 generations. Thirty trial runs were carried out for each experiment. Generational replacement was used with an elite size of 2. A one-point crossover operator was used in conjunction with standard integer mutation.

4.2 Fitness Function

The fitness function is what makes selection possible for an EA. A poorly chosen fitness function means that the algorithm will generate good fitness values but poor results, or in this case, poor behaviour. We have published

Table 1: Configuration values for Toribash Joints, values 3 and 4 always represent hold and relax respectively.

Number	Joint	Contract/Left/Raise	Extend/Right/Lower
0	neck	2	1
1	chest	2	1
2	lumbar	2	1
3	abdomen	1	2
4	right pectoral	2	1
5	right shoulder	2	1
6	right elbow	2	1
7	left pectoral	2	1
8	left shoulder	2	1
9	left elbow	2	1
10	right wrist	2	1
11	left wrist	2	1
12	right glute	1	2
13	left glute	1	2
14	right hip	1	2
15	left hip	1	2
16	right knee	2	1
17	left knee	2	1
18	right ankle	1	2
19	left ankle	1	2
20	left hand	NA	NA
21	right hand	NA	NA

our code for the GA on the Toribash forums and this has resulted in a large amount of user feedback about the fitness function [3]. There has also been significant development of different fitness functions for the GA. Despite this input, we are using the most basic fitness function for the experiment. The reason is that we want to reduce the confounding effect of additional fitness parameters and so that it may be used as a baseline for future fitness function development. The fitness function we use for the algorithm is the score Tori obtains minus the score Uke obtains. If a player was disqualified then their fitness was set to zero.

4.3 Developing Opening Moves

In this experiment, we compared differing rates of mutation and crossover when applied to both the standard game mod and the katana sword mod. One of the goals of this experiment is to discover if there is a truly 'optimal' move in a static environment with only one time step. The different experimental settings are shown in Table 2. The results obtained were compared against randomly created individuals. Elitism was used to store the best randomly generated individual over a run.

Table 2: Experimental settings

Setting	Mutation Rate	Crossover rate
Standard	10%	70%
Low Mutation	1%	70%
No Crossover	10%	0%

4.3.1 Normal Game Mode Results

The results for the experiment carried out using the normal game mode are shown in Figures 3 and 4. A one-way ANOVA test was performed to test statistical significance of the results. The results showed that low mutation with crossover performed no better than randomly generated individuals. While standard mutation with crossover clearly outperformed randomly generated individuals, it did not outperform the algorithm that used mutation exclusively. These results would suggest that standard one-point crossover on the current move representation does not have a beneficial effect. Figure 4 shows that low mutation with crossover is less destructive on the general population as they have a statistically significant higher average but this did not translate into better individuals being evolved. At this point we investigated what moves the Tori was actually making.

Samples of Toribash behaviour may be viewed on the NCRA website [1]. The large majority where either head grapples or direct blows to the head, as shown in Figure 5. This shows that for a single move in the standard game mode, there is indeed an optimal move. It also suggests a points multiplier for blows to the head, a factor that could be taken into account in the fitness function. These results show that the GA was capable of evolving fighting behaviours better than a random search but also that the effect of crossover was minimal.

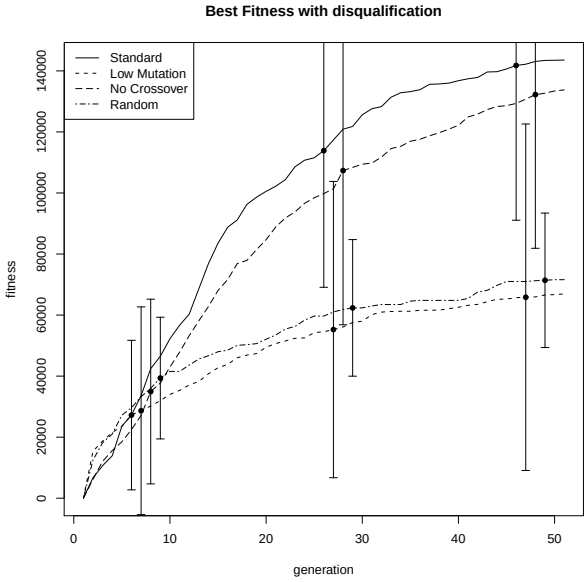

Figure 3: Best Fitness for a Single Move

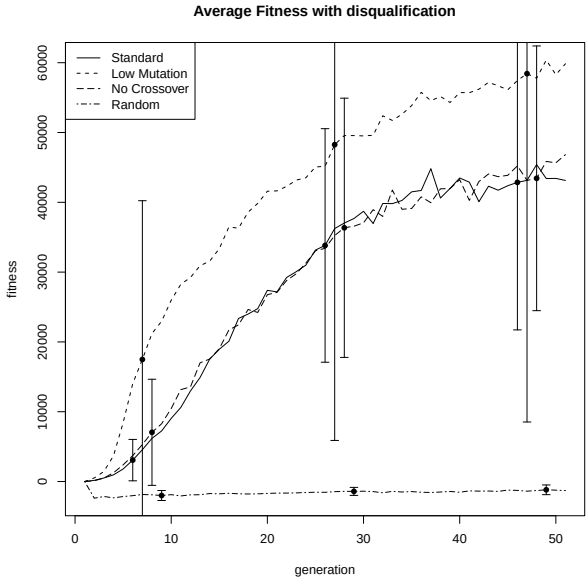

Figure 4: Average Fitness for a Single Move

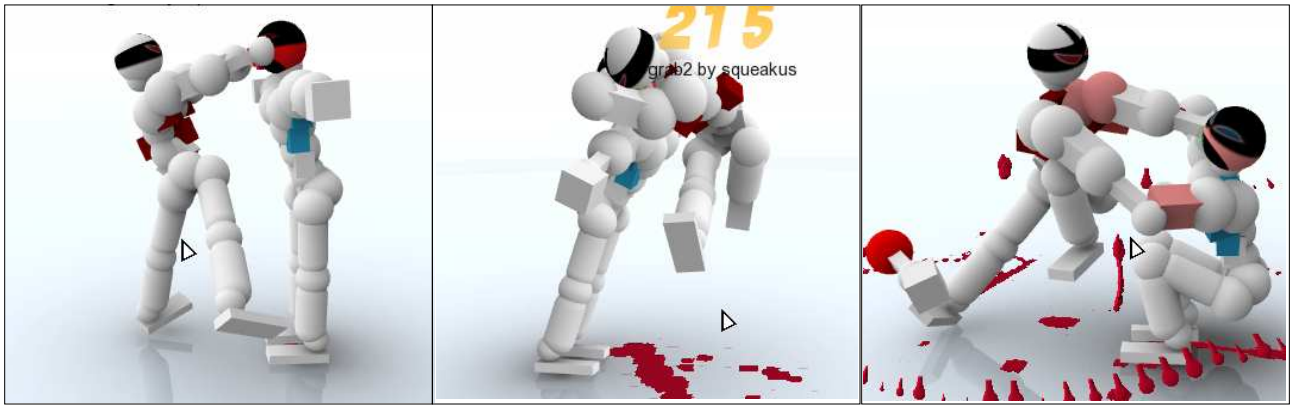

Figure 5: Examples from Standard Moves

4.3.2 Sword Game Mode Results

The results for the experiment carried out using the sword game mode are shown in Figures 6 and 7. The results were similar to the results for the normal game mode. The one-way ANOVA again showed that standard and no crossover were statistically similar and that low mutation performed similarly to random, although the addition of weapons has dramatically increased both the score and the variance. Upon investigation of the moves Tori was making, a far greater range of attacks were found than in the standard game mode. Every best move generated involved the Tori using the sword. This shows that the GA is capable of learning to use objects in the environment. While there was no single dominant move, the moves predominantly cut either vertically through or across the midriff, thus maximising the number of joints severed, as shown in Figure 8. In conclusion, the GA was capable of using objects in the environment and adjusting its strategy to accommodate these objects.

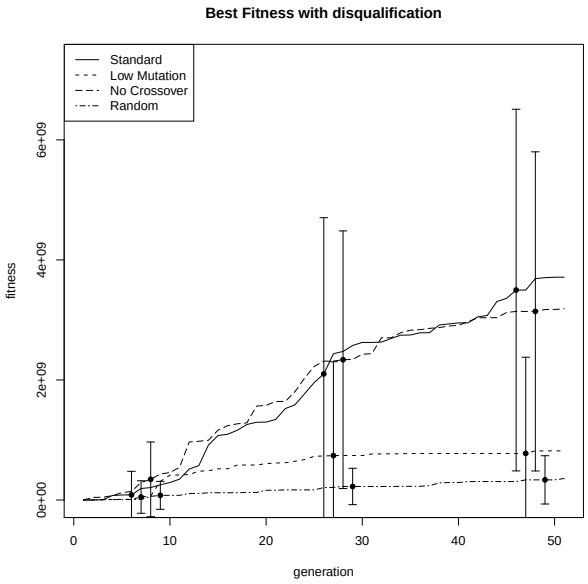

Figure 6: Best Fitness with Sword

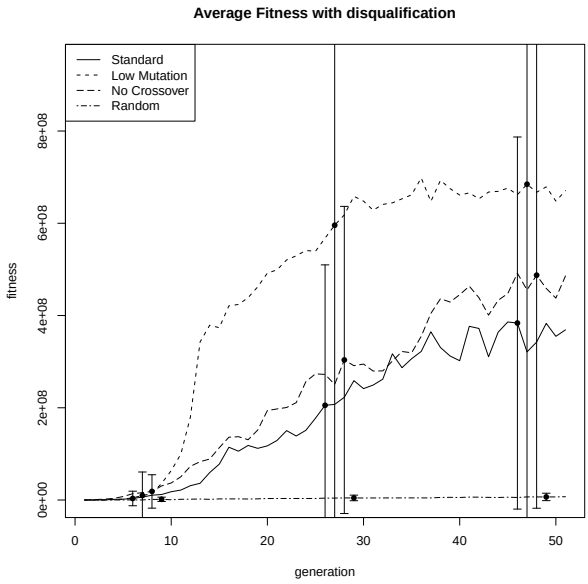

Figure 7: Average Fitness with Sword

Figure 8: Example of Sword Moves

4.4 Developing Move Combinations

In this section we investigate whether it is possible for a GA to evolve a coherent sequence of moves. To accomplish this goal we increased the size of the chromosome so that it could encode for the new moves. We allowed the GA to evolve 3 move combinations. The experiments were carried out with the same settings as shown in Table 2. As we have increased the chromosome size to accommodate additional moves, the application of single point crossover might not be adequate to explore the search space. To take advantage of the move representation and increase the crossover rate we devised new crossover operators. This section describes in detail the operation of three new crossover operators we created for evolving move combinations.

4.4.1 Multiple Point Crossover

The multiple point crossover operator selects a point within each move of an individual and then exchanges the codons up to that point with the other individual. The crossover probability is used to calculate the likelihood of whether each point is swapped. The repeated application of the probability means there is an increase that more than one crossover event will occur.

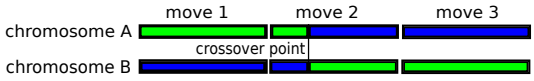

Figure 9: Single point crossover

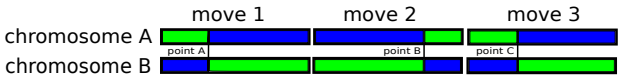

Figure 10: multiple point crossover

4.4.2 Move Preserving Crossover

Move preserving crossover is similar to multiple point crossover except that the points are always on the boundary of a move. By swapping whole moves, this operator maintains the integrity of contiguous blocks of codons. The moves are always swapped with moves of the same sequential order. This operator uses the same crossover probability technique as multiple point crossover so the is an increase in the probability of a single mutation event occurring

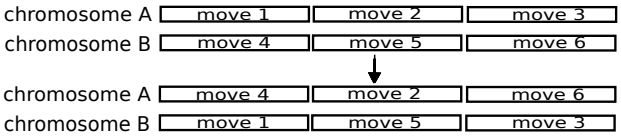

Figure 11: Move preserving crossover

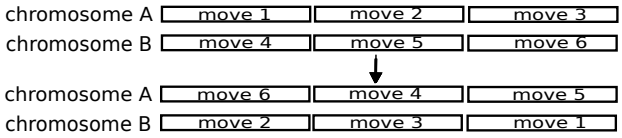

Figure 12: Move shuffling crossover

4.4.3 Move Shuffling Crossover

Move shuffling crossover takes all the moves in their entirety, reorders them and then recombines them to create new combinations from the moves. Unlike the previous operators, it is an all or nothing approach. It uses the crossover probability to decided whether to shuffle the moves or not.

4.5 Move Combination Results

The results for the different crossover operators are shown in Figures 13 and 14. The graphs show a comparison between a single move and the different crossover operators. A one-way ANOVA was performed on the data and it showed that allowing for move combinations added a statistically significant (albeit very small) improvement to the overall fitness. As regards the crossover operators, one-point crossover and move preserving crossover performed as well as each other, this was followed by multiple point crossover. Move shuffling crossover performed the worst, with results only slightly better than randomly generated results. The average fitness for multiple point and move shuffling crossover is far below that of the others which implies that these crossover operators had a destructive effect on the population.

These results would lead us to conclude that the additional moves did not add significant benefit but this is true only as regards the fitness function. One of the ways we can examine if it uses the additional chromosomes is by looking at the moves themselves. What we discovered is that approximately 60% of the moves involve a direct hit or grapple followed by additional blows [1], but in several instances it is clear the Tori is using the additional moves to his advantage. We have added links to videos showing several examples of this behaviour [1]. This evidence would lead us to believe that if we selected for move chaining in the fitness function then we could obtain this behaviour more readily.

5 Discussion

Our experiments showed that an evolutionary algorithm was capable of evolving good results for Toribash but also showed that crossover had either a negligible or destructive effect. This could have been caused by the representation we chose for the moves. We used the given ordering for moves as shown in Table 1. With the exception of the pectoral,shoulder and elbow grouping, the limb order moves from the top to the bottom of Tori. As groups of limbs were not ordered sequentially in the chromosome, this meant that there was no clearly defined 'building blocks'. The effect of crossover could be dramatically improved if it was possible to choose a better representation. We also hope to examine if better results can be obtained by using simpler methods such as a hill climbing algorithm or one that uses mutation exclusively, such as an evolutionary strategy.

6 Conclusion & Future work

In this paper, we set out to examine whether a Genetic algorithm was capable of evolving moves for a selection of game modes in Toribash. Our results showed that the GA was capable of evolving behaviour for both the default game mode and the katana sword mod. While crossover did not have a significant impact, our results showed it greatly outperformed random search. We continued our investigation by allowing additional moves and creating specialised crossover operators for multiple move evolution. We found that despite the overall result only being a slight improvement on single moves, that there were several instances where Tori used the additional moves to his advantage. We conclude that the genetic algorithm has shown that it is possible to evolve

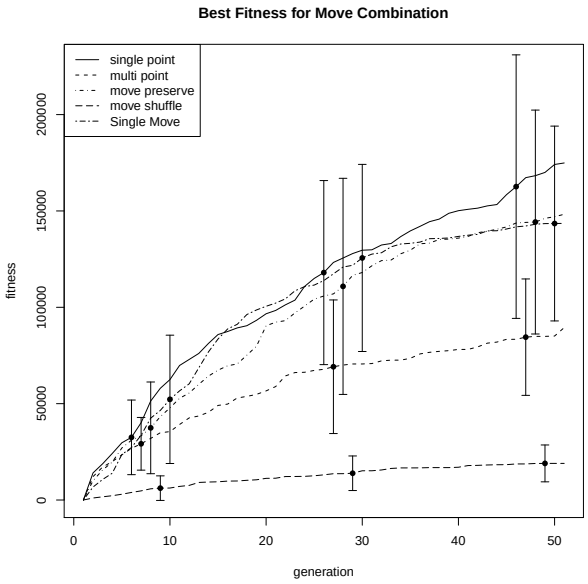

Figure 13: Best Fitness for a Move Combination

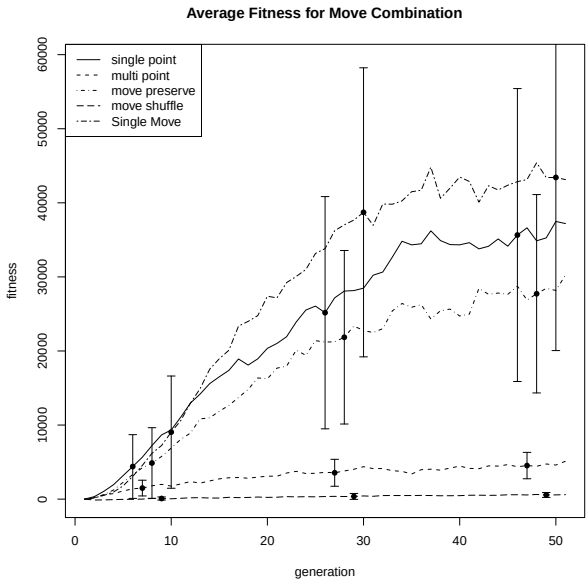

Figure 14: Avr. Fitness for a Move Combination

complex behaviours for Toribash. In our future work, we will compare our current approach with other search methods such as hill climbing and EDAs. We intend to develop the move representation so that crossover has a beneficial impact. Once this has been accomplished we hope to use Toribash as a testbed for the coevolution of bots. This will allow us to compare different machine learning techniques and examine the generic qualities of their behaviours. We can then test the bots on-line to examine if they are truly comparable to human playing.

7 Acknowledgments

We would like to thank Keith Begley for drawing our attention to this problem, Erik Hemberg for his invaluable feedback on this paper and Aidan Molloy for his unceasing help and support. We would also like to thank our funding agency, Science Foundation Ireland. This research is based upon works supported by the Science Foundation Ireland under Grant No. 08/IN.1/I1868.

References

[1] Link to videos demonstrating Toribash behaviour. `http://ncra.ucd.ie/members/byrnej.html`.

[2] Toribash. `http://www.toribash.com`.

[3] Toribash GA. `http://forums.toribash.com/showthread.php?t=17010`.

[4] M. Buckland and P. Publishing. *AI techniques for game programming*. Course Technology, 2002.

[5] N. Cole, S. Louis, and C. Miles. Using a genetic algorithm to tune first-person shooter bots. In *Proceedings of the International Congress on Evolutionary Computation*, volume 1, pages 139–145, 2004.

[6] Darren Doherty and Colm O'Riordan. Effects of shared perception on the evolution of squad behaviours. *IEEE Transactions on Computational Intelligence and Games*, 1(1):50–62, march 2009.

[7] S. Grand and D. Cliff. Creatures: Entertainment software agents with artificial life. *Autonomous Agents and Multi-Agent Systems*, 1(1):39–57, 1998.

[8] E.J. Hastings, R.K. Guha, and K.O. Stanley. Evolving content in the galactic arms race video game. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2009.

[9] P. Molyneux. Postmortem: Lionhead Studios Black & White. *Game Developer*, 2001.

[10] Steve Rabin. *AI Game Programming Wisdom*. Charles River Media, Inc., Rockland, MA, USA, 2002.