Finding an optimal Noughts and Crosses strategy using a Genetic Algorithm

Harper Ford

hf304@exeter.ac.uk

ABSTRACT

Noughts and Crosses is a game that dates back to 1300BC and is played globally by children and adults alike as a form of simple entertainment. In this paper we review and build on previous Genetic Algorithm (GA) implementations aimed at producing a perfect "no-loss" Noughts and Crosses strategy. The main flaw outlined from previous approaches is that the fitness functions used values draws and wins equally. This decision leads to solutions that often throw easy wins in exchange for a draw. We implement an alternate fitness function definition that maximizes a solutions desire to win. We end by showing our proposal can consistently find win greedy "no-lose" strategies when starting the game first.

KEYWORDS

Evolutionary Computation; Genetic Algorithms; Noughts & Crosses; Tic-Tac-Toe; Fitness Functions.

1. INTRODUCTION

Noughts and Crosses is a 2-player game played on a 3x3 grid. Players are assigned a symbol (either "X" or "O"). Starting with "X" the players take turns placing their symbol into empty cells on the grid. The game is won when a player connects 3 of their symbols either horizontally, vertically, or diagonally. The game ends in a draw if there is currently no winner and no free spaces to place a new symbol.

Due to the fact a single game of Noughts and Crosses has only 9 possible moves; playing as "X" has a significant advantage by going first, the player can instantly attack and by the end of their second turn be threatening victory. Therefore, the requisite lies upon the player assigned "O" to not lose rather than to win. In a hypothetical Noughts and Crosses tournament where:

- Players play each other an even number of times.
- Players alternate symbols each game.
- A win/draw/lose is valued as 1/0.5/0 points, respectively.
- The player with the largest total of points after their games is the victor.

A necessary strategy to win this tournament is to try and win all the games you play as "X" and draw all the games you start as "O" (Since winning as "O" is difficult to accomplish with reasonable counterplay from "X" and drawing as "X" can put you behind deeply in the tournament). This strategy has been incorrectly summarised by some as a "no-lose" strategy where an optimal player only wins or draws and never loses [1, 2]. The flaw in this logic lies in the fact that an optimal player by this standard is happy to draw when they could have won meaning their strategy when playing as "X" is flawed if they partook in the tournament.

Ultimately a strategy needs devising that winning over drawing and drawing over losing.

A Genetic Algorithm (GA) is a population-based optimisation technique that relies on; a proper problem formulation; fitness function; crossover, and mutation operators to iteratively find and assess potential solutions to an optimisation problem. An in-depth explanation of GA's can be found here [4]. GA has been used to successfully create "no-lose" Noughts and Crosses strategies [1, 2]. GA has also been used to create a supposed win greedy" no-lose" strategy [3].

In the following section we review and discuss the literature surrounding Noughts and Crosses strategy optimisation using a GA.

2. LITERATURE REVIEW

The key areas from the literature we focus on are problem formulations, fitness function definitions, replication, and crossover operators. Each concept used is briefly broken down and its pros and cons outlined.

2.1 Problem Formulation

The general consensus found was that applying a GA to a Noughts and Crosses strategy problem is relatively simple when using a *genome-model* GA, that is; a potential solution can be described as a set of strings with associated values (much like a gene and allele relationship). The full formulation suggested and implemented can be found down in ***Section 3*** but a brief outline follows: Each legitimate and unique Noughts and Crosses game-state 3x3 grid is flattened into a string and acts as a gene. The allele associated with each gene is the grid position to place your symbol "X" or "O". There are some slight intricacies to this process in order to decrease runtime and they are also laid out in ***Section 3***. This formulation is all we could find regarding Noughts and Crosses strategy optimization using a GA and it has been proven to work well [1, 2, 3].

2.2 Fitness Function

Evaluation of a populations members is an important step for any GA since it decides which members to crossover or replicate for future generations, potentially leading to an improved next generation. Given the problem of creating an optimal Noughts and Crosses strategy, when deciding how to evaluate strategies one must start by calculating what an optimal strategy would look like.

For Noughts and Crosses an optimal solution changes depending on certain factors, the biggest of which is whether you are trying to optimize for wins or solely trying to remove loses. Finding a "no-lose" strategy has been implemented by G. Hochmuth [1] and A. Bhatt et al. [2]. The fitness functions proposed have each member in the population exhaustively play all possible games they could encounter leading to a large and branching evaluation of each member. A member's fitness is then described by either of the following:

1) $$f(x_i) = \frac{n_{total}[x_i] - n_{lost}[x_i]}{n_{total}[x_i]} \text{ [1]}$$

2) $$f(x_i) = \frac{n_{lost}[x_i]}{n_{total}[x_i]} \text{ [2]}$$

Figure 1. Different member fitness evaluation techniques.

Each member of the population plays as both "X" and then "O" to allow for an all-around strategy to be developed because both symbols have different situations they could come across. This evaluation technique increases runtime drastically as each member must play 10 branching games (1 as "X" and 9 as "O" where the opponent plays a different starting move each time).

It was shown that optimizing a member's strategy as "X" to not lose happened within 10 iterations whereas optimizing for "O" took 432 iterations at worst [2]. The difference in convergence times can be easily explained by the inherent advantage going first and playing as "X" gives you, since the number of moves you can play that lead to a loss are far smaller than playing second as "O". The fitness functions implemented by G. Hochmuth [1] and A. Bhatt et al. [2] achieve what their respective papers intended, to create a "no-lose" strategy for Noughts and Crosses, the issue comes with trying to develop a strategy that wants to win. A fitness evaluation was proposed that weighted wins and draws differently attempting to create a win greedy "no-lose" strategy but that was found to reach local maxima stagnation [1]. The reasoning for this is not explained in the paper but is more than likely a result of the size and inter-dependence of a member's genome. A positive mutation may not be valued correctly in future generations as other genes that are reached later on in the fitness evaluation are not yet optimised resulting in a very poor score for the positive mutation. This would lead to local maxima stagnation as exploration is punished because not enough mutations are occurring to fully alter a strategy in a positive direction. Another way of putting this phenomenon is that because future moves cannot be observed or predicted, logical mutations struggle to survive.

The main flaw with the working fitness functions described above is that they effectively attribute a win and a draw as equivalent giving generations no incentive to improve their total number of wins, this leads to solutions that play for draws even in positions they can win. A. Bhatt et al. [2] did however track the win-draw ratios of any solutions they found and during their trails did find an optimal win greedy "no-lose" strategy, the downside is they came by this solution by happenstance and were not directly searching for it.

A greedier win hungry fitness function was proposed by [3], this had a heavy negative weighting for a loss, a large positive weighting for a win and a neutral weighting for a draw. Members of a population played 1000 random games and had their fitness score evaluated as a sum of the above weightings. The random games result in an uncertainty if not ran for 1000s of iterations since there is no guarantee that the member is being evaluated on every crucial move. [3] found success in creating a competitive Noughts and Crosses strategy though there was no way to verify the actual performance in other fitness measures.

2.3 Replication, Crossover, and Mutation

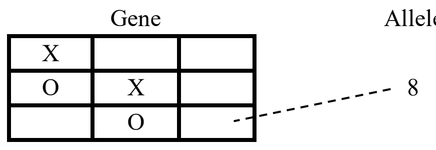
A final crucial step to any GA is how the next generation is formed and influenced by previous generations using replication, crossover, and mutation. G. Hochmuth [1] and A. Bhatt et al. [2] implement multi point crossovers from the best two members from a generation, the crossover produces a new member that is a combination of the two parents and is passed into a new generation, alongside newly generated random solutions. This keeps exploration of the solution space high because a large number of new solutions are constantly tested generation to generation. [3] proposes a method that is greedier and more driven for exploitation of solutions that performed well. [3] takes a selection of the best performing solutions from a given population and creates the entire next population from this selection using mutation only to explore the solution space, this approach could potentially suffer from premature convergence if the rate of mutations is not high enough [4]. G. Hochmuth [1] and A. Bhatt et al. [2] implemented a standard mutation procedure in which a child solution had a chance of some of its alleles being randomized. A lower mutation rate was found to perform best. [3] proposed only a single gene per child solution should be mutated but attested to the fact that this was probably too low of a mutation rate to see significant exploration.

3. DESIGN

3.1 Individuals and Strategy Encoding

A brief explanation of the strategy encoding we use is laid out in this section following the advice from the literature [1, 2, 3].

A Noughts and Crosses strategy is represented as a table of relationships between all possible game-states and a corresponding move to make. Applying this formulation to a GA is rather simple if we employ a genome model. A genome-based GA is proposed where each gene is represented by a unique game-state and the associated allele is the next move to make.



Gene

2	1	1
3	2	1
1	3	1

Gene
211321131

Figure 2. An example gene-allele encoding.

The gene encoding is collapsed into a string where the symbols are replaced by integers following this pattern: {“ “ : 1, “X” : 2, “O” : 3}. The allele is the index to play the next move in, an example of this {gene: allele} relationship would be {131121112: 0}.
There are 3^9 or 19,683 ways of filling a 3x3 grid with 3 different symbols (“X”/”O”/” “) [5]. In the realm of Noughts and Crosses only some of these permutations can legitimately occur in a single game. From the 19,683 possible game states we remove all those that are illegal given the rules of Noughts and Crosses [5].
After removing the illegitimate states there are 765 unique game-states that can be occur in Noughts and Crosses [5]. Unique means that all game situations that are identical by either rotation or mirroring count as a single game situation. For a single game-state there are 7 duplicates by this standard. We keep one duplicate to be used as a gene, we call this the base-case game-state. Lastly, the 765 game-states includes games that have already ended in either a draw or win by a player. These end-game states are also removed from the gene pool to reduce the size of the problem while not affecting the algorithms integrity. An example solution therefore has 627 genes (one for each potential game-state they could ever encounter) and 627 associated alleles that refer to which position in the grid the member should play their next move (0-8).

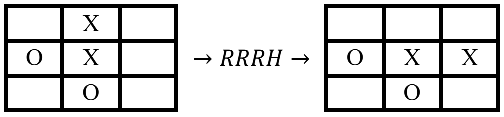


Figure 3. The mapping of one game-state to another

A member plays a game by being shown a state and the corresponding gene is found, any rotations or horizontal mirroring (indicated as R and H respectively) that is required to translate the input state to the gene is performed and recorded. The members symbol (“X” or “O”, this depends on if the member is going first or not) is then placed in the state at the position indicated by the *allele*, the state then reverses all performed transformations (rotations or mirroring) and play continues.
A hashing function has been constructed that maps any game-state to its base-case, similar functionalities are proposed in [1, 2, 3] but no pseudocode is provided.
The hash first finds the determinant of the 3x3 grid using the same symbol to integer encoding described earlier, {“ “:1, “X”:2, “O”:3}. Next, each row and column have their contents multiplied and are summed together. The same procedure is done for the grids diagonal lines too. Finally, the procedure is repeated for the center row and column. The entire grid is then scaled by the ***Heat*** variable (a 3x3 matrix) and all cells are summed together. The outputted hash is a combination of all the above calculations into a single string.

Algorithm 1 Hashing a game-state to its base-case

```
procedure HASH(game-state){
  A ← Find the game-state determinant
  B ← Sum the products of the game-state rows and
  columns
  C ←Sum the products of the game-state diagonals
  D ←Sum the products of the game-state center row and
  column

  Heat ← 3x3 matrix [3,5,3
                     5,8,5
                     3,5,3]
  Map ← empty 3x3 matrix

  for cell in Heat do
      $Map_{cell}$ ← $game\text{-}state_{cell} \cdot Heat_{cell}$

  E ← Sum all the values in Map

  return A:B:C:D:E
}
```

Algorithm 1. Maps identical game-states to the same hash, this is used to quickly find the gene associated with any game-state no matter what the rotation or horizontal mirroring.

State

	X	
O	X	
	O	

Hash

2:176:4:18:73

State

O	X	X
	O	

Hash

2:176:4:18:73

The above steps have been implemented in order to speed up any game-state to base-case calculations, since so many lookups occur per single game it is not viable to algorithmically find the base-case gene through manual rotation and horizontal mirroring.

3.2 Genetic Algorithm

The GA proposed in this paper uses a *genome-model* and employs the solution encoding described in **3.1** to create Noughts and Crosses strategies, an optimal strategy is one that capitalizes on potential wins and never loses. The strategies produced using this technique are unique in that they are better than pure “no-lose” strategies. The following sections breakdown and explain the reasonings behind the inner workings of the GA we propose.

3.2.1 Fitness Evaluation

The fitness evaluation used here is reminiscent of the first attempt formulated by G. Hochmuth [1]. The evaluation was found to lead to local maxima stagnation because of what we assume is premature convergence due to the low mutation rate used (0.001) on a large genome with strong interdependence between genes. The fitness evaluation we implement is as follows:

- Each member plays all possible games, as both “X” and “O”.
- A win/draw/loss nets the member different points, the point values used can be found in ***Section 4.1.1*** & ***4.2.1***.
- The fitness of a member is their sum of points after playing every possible game.

3.2.2 Replication, Crossover, and Mutation

Our replication process is as follows; the best in the last generation is passed into the next, called elite perseverance. We also select the top N solutions of a population and replicate them into the next generation whilst applying a mutation procedure to them.

To help prevent premature convergence we also select N random solutions from the previous generation to be replicated with mutations into the next generation, this technique is comparable to the Stochastic Uniform Selection (SUS) process suggested by A. Bhatt [2].

We do not use any crossover, solely relying on mutations to improve generation to generation. This has been done to better exploit high fitness solutions [3]. To increase exploration, we fill the remainder of the population with random solutions.

The mutation procedure implemented runs through a solution making mutations; each allele has a static *p_muta* chance of mutating to a random valid value. Our approach is similar to that found in [3] where the entire next generation are the children of the best performing from the last generation, the difference is that only some of the new generation are children with our approach, the others are children from randomly sampled solutions from the last population alongside new random solutions, this re-expands the exploration space that was reduced in [3].

3.2.3 Summary

Objective	Find a win greedy “no-lose” Noughts & Crosses strategy.
Structure	Fixed-length string of ; Alphabet size of 9 (index of the move to take); Mapping occurs from game situation to move to take.
Fitness Cases	An individual that does not lose and strives to win over drawing.
Raw Fitness	A weighted score based on win/draw/lose counts.
Operators Defined	Elitist and random replication, Mutation
Population Initialization	Populations are initialized randomly; no seeding heuristics are applied.

Table 1. Implemented Genetic Algorithm Formulation

4. EXPERIMENTATION

We can split the overall strategy of an optimal solution into a strategy when playing as “X” and a strategy when playing as” O”. This split is useful since it allows us to make changes to our parameters and approaches for each strategy, it also makes the problem easier to optimize for since it splits a complex fitness landscape into 2 less complex ones (an “X” genome with size {} and an “O” genome with size {}). After finding optimal solutions to both sides, we can merge the solutions into a fully optimal Noughts and Crosses strategy since the game-states “X” and “O” optimize for are different and therefore will not clash.

The layout of this section is as follows: The results for our implementation when optimizing for a win greedy “no-lose” strategy for “X”, followed by our attempt for “O” and ending with an implementation of G. Hochmuth’s [1] proposal. The section ends with an overview analysis of all results.

4.1 Strategy “X”

4.1.1 Parameter Tuning

Win	Draw	Loss
1	-15	-100

Figure 4. The points awarded for each end-game state reached when starting as “X”.

The points system used to evaluate an “X” playing solutions fitness is shown above. A win is weighted higher than a draw which is weighted higher than a loss. The draw weighting when playing as “X” cannot be 0 as this leads to solutions with many draws scoring too high and being reproduced into the next generation. There is, however, reason for concern when imaging a solution that draws a lot scoring overall far lower than a solution that does not draw that much but loses a few games; the weighting in **fig 4** works since there are fewer ways an “X” player can draw so there is little concern with a solutions overall draw score being lower than their loss score.

It was found that a mutation rate far higher than 0.001 is needed to stop the stagnation found by G. Hochmuth [1] when using a weighting system to score solutions as described above. This is because sequences of genes often need to change in order to see any fitness score improvements thanks to the interdependence between genes. A mutation rate of **0.1** was found to allow for the best ratio of exploration to exploitation, a whole 100 times larger than recommended.

4.1.2 Results

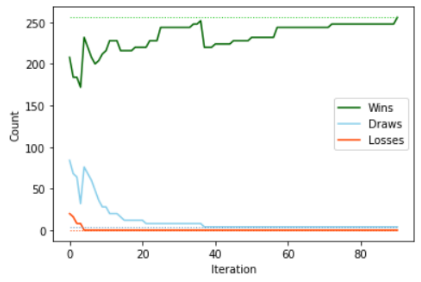


Figure 5. Win/Draw/Loss Counts for “X”.

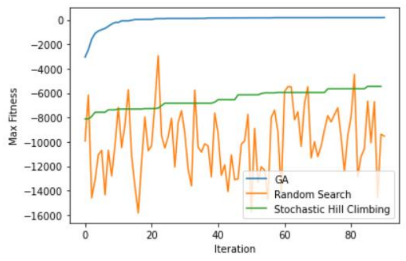


Figure 6. The maximum fitness per generation finding an optimal win greedy "no-lose" "X" strategy.

Fig 5 and **fig 6** were instantiated with a population size = 100, mutation rate = 0.1, elite preservation = 1, random selection = 20 and the top 5 solutions were selected to be mutated and passed into the next generation 7 times each. It was found that for an "X" strategy optimisation the fitness evaluation prescribed reached a maximum of 196, scoring 260 wins, 4 draws, and 0 losses. This raises confusion with the findings that the best win-to-draw ratio possible was 41:3 [2]. Achieving 3 draws is not possible with "X" starting in the center cell (which our strategy opted for) and playing all possible games, this is because a draw can be conceded by "O" playing in 1 of the 4 corners on their first turn, resulting in a minimum of 4 draws.

You can see from **fig 5** that even after the loss and draw counts plateau there are steady increases in wins, this is due to the GA finding solutions that provide more ways to win by extending games in positions they cannot lose or draw in.

4.2 Strategy "O"

4.2.1 Parameter Tuning

Win	Draw	Loss
1	0	-100

Figure 7. The points awarded for each end-game state reached when starting as "O".

The concerns found with the weightings for a strategy starting as "X" do not apply to the strategy that starts as "O" since draws are far more likely and therefore cannot be negatively weighted as you risk promoting solutions that lose since they would score better than solutions with lots of draws even though drawing is a far more desirable outcome.

The mutation rate does not require changing from the tuning for optimizing for an "X" strategy; a variety of rates were tried but none prevented sub-optimal stagnation.

4.2.2 Results

Fig 8 and **fig 9** show that optimizing a strategy for "O" stagnated at around 150 iterations (multiple runs were attempted and stagnation occurred in all at some point). The run ended with 640 wins, 128 draws, and 6 loses. These results are promising as the strategy developed clearly has a preference of winning over drawing but still loses a small number of games. The reasoning for this stagnation can be found below. Even after attempting to complete the optimisation with a Stochastic Hill Climber the score remained stagnant (**fig 10)**.

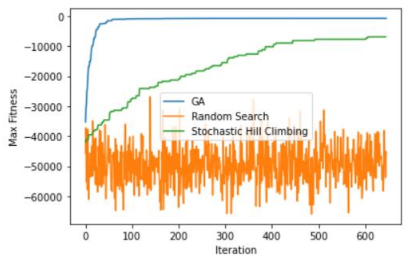


Figure 8. The maximum fitness per generation trying to find an optimal win greedy "no-lose" "O" strategy.

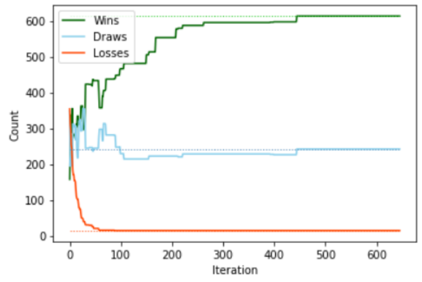


Figure 9. Win/Draw/Loss Counts for "O".

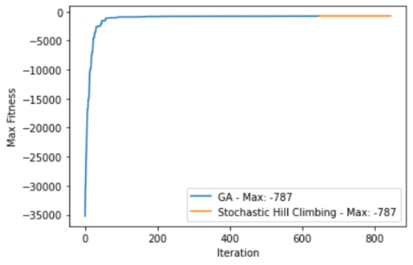


Figure 10. Using a Stochastic Hill Climber to improve "O".

4.3 "No-lose" Strategy

This section breaks down the results from implementing the GA proposed by G. Hochmuth [1]. The aim for their research was to create a "no-lose" Noughts and Crosses strategy for both "X" and "O".

4.3.1 Parameter Tuning

Mutation Rate	0.001 per allele
Multi-point Crossover Rate	0.15
Replication Rate	0.1

Table 2. The parameter settings used by G. Hochmuth [1].

4.3.2 Results

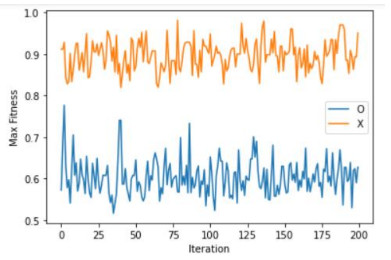


Figure 11. G. Hochmuth's [1] GA

Using the proposed GA from G. Hochmuth [1] there seems to be no progression between generations. This is probably due to the extremely low mutation rate and the fact that only 2 solutions from each generation were used for crossover, these issues fail to properly exploit potentially good solutions. How G. Hochmuth [1] found compelling results in their paper is confusing. A. Bhatt et al. implement an SUS combined with elite preservation to stop sub-optimal convergence whilst exploiting good solutions; both these techniques are very similar to the replication process we propose hence why our approach can find optimal "X" strategies consistently [2].

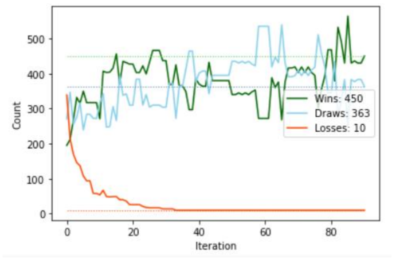


Figure 12. Stagnated results for "O".

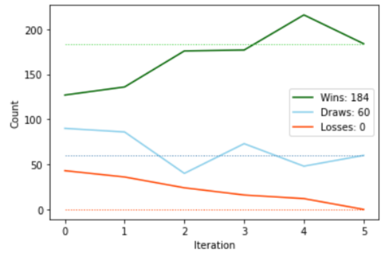


Figure 13. "No-lose" "X" strategy.

Employing our replication, mutation and crossover operators proposed in this paper alongside the fitness evaluation used by G. Hochmuth [1] we can easily evolve a "no-lose" strategy for "X" but not for "O". We see that the strategy for "X" eliminates all loses around 4 iterations. This is in keeping with the findings from A. Bhatt et al. who discovered "X" could remove all loses within 4-10 iterations [2]. The strategy for "O" takes considerably more iterations to eliminate a majority of loses, this is due to the fact "X" has such an advantage and imposes more loss chances onto "O". The strategy developed for "X" has a bad ratio of wins-draws-loses compared to the "X" strategies optimizing using our fitness evaluation, this is the flaw with G. Hochmuth's [1] approach; the strategy developed cannot lose but struggles to win.

4.4 Analysis

In our attempt to find a win greedy "no-lose" strategy for "X" using the fitness evaluation described in ***Section 3.2.1*** we see a dramatic increase in maximum generation fitness within the first 20 iterations as loses are removed. We then see a struggle to fully optimize the strategy for wins over draws. The maximum fitness of 196 is reached in 98 iterations for "X". This struggle occurs for the a few reasons: **1)** The interdependence between genes. **2)** GA has a large weakness with final convergence, especially for large genomes such as the one described here. Gene interdependence can be described as the issue that a very specific series of moves are required to win a game and to not draw or lose it. Since a single gene mutation (even if logically positive) relies on other genes to increase the members fitness score we need to wait for a sequence of moves to be randomly mutated in order to see any fitness score improvements, this issue has been overcome by increasing the mutation rate which allows for more chance that a sequence of moves is mutated at the same time. The GA weakness could be solved by solely using the GA to eliminate terrible solutions and then complete the necessary final convergence using an Ant-Colony-Optimisation technique (which has been shown to be good at final convergence) [7, 8].

For our "O" strategy optimisation we found solutions got stuck at a local maximum when using the fitness scoring system prescribed, this was also found by G. Hochmuth [1] who stated they stepped away from weighting wins, draws, and loses for the same reason. It seems that "O" struggles with eliminating the final few losses and draws and could be improved by mixing with an ACO technique once stagnation is reached [7, 8], similar to our attempt to exploit a Stochastic Hill Climber that unfortunately failed to improve solutions (**fig 10**).

5. CONCLUSION

Noughts and Crosses is a simple game that has different strategies depending on what symbol you are playing as, applying the problem of finding an optimal strategy to a GA is an easily doable task but the parameter tuning to find optimal solutions for both "X" and "O" require different considerations. It was found that not only was optimizing loses out of the "X" strategy far simpler than for "O" but optimizing for wins over draws was far simpler. Trying to find the optimal win greedy "no-lose" strategy for "O" was a far more taxing computation process with many ways the algorithm could either prematurely converge or get stuck in local maximums. This paper has proposed and implemented GA that reliably finds optimal win greedy "no-lose" strategies for playing as "X", something not achieved by the reviewed literature, we do however fail to find a pure "no-lose" strategy for "O" by following the work of G. Hochmuth [1]. Of course, we are disappointed that an optimal win greedy "no-lose" strategy could not be reliably found for "O" and will continue to research in an attempt to rectify this.

The main issue with applying the Noughts and Crosses genome formulation is that sequences of good moves are required for a perfect strategy, the problem is far more manageable by an algorithm that has good look ahead evaluations such as *Deep-Reinforcement Learning* (DRL) [6]. GA's weaknesses came to light with our implementation that being **1)** sequences of good moves were hard to find **2)** Final convergence was hard to accomplish due to the large genome size. The sequence issue could potentially be solved by adapting the multi-point crossover functions proposed [1, 2]. The adaptation would be to use a fixed-point crossover where the points split up different sequential genes (that is not adjacent in the genome but sequential in games). This change would allow crossover to mix sequences of moves instead of random selections of gene values that are not related. The final convergence issue could be solved by using the GA to find a group of good solutions and then applying an ACO technique to refine the solutions, as they have been proven to perform well for final convergence [7, 8].

REFERENCES

[1] G. Hochmuth. On the genetic evolution of a perfect Tic-tac-toe strategy, pp. 75–82, Stanford University Book Store (http://www.geneticprogramming.org/studentpapers2003.html, 2003), Accessed: 05/21.

[2] A. Bhatt et al. Evolution of No-loss Strategies for the Game of Tic-Tac-Toe, Kanpur Genetic Algorithms Laboratory (KanGAL), Report Number 2007002.

[3] Unknown Author. An invincible Genetic Algorithm, TropicalCoder, (http://www.tropicalcoder.com/GeneticAlgorithm.htm), Accessed: 05/21

[4] D. E. Goldberg. Genetic Algorithms for Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989.

[5] Mathrec contributors. Mathematical recreations, repository for mathematical diversions, (http://www.mathrec.org/old/2002jan/solutions.htm, 2002), Accessed on TheWayBackMachine (https://web.archive.org/web/20161023060942/http://www.mathrec.org/old/2002jan/solutions.html), Accessed: 05/21.

[6] M. van de Steeg et al. Temporal Difference Learning for the Game Tic-Tac-Toe 3D: Applying Structure to Neural Networks, 2015 IEEE Symposium Series on Computational Intelligence, 2015.

[7] W. J. Gutjahr. ACO Algorithms with Guaranteed Convergence to the Optimal Solution, Department of Statistics and Decision Support System, University of Vienna.

[8] H. Duan. Ant Colony Optimization: Principle, Convergence and Application, Handbook of Swarm Intelligence, ALO 8, pp. 373-388, Springer-Verlag Berlin Heidelberg, 2011.