Comparison of Genetic Algorithm Training Methods as Applied to Tic-Tac-Toe

Vash Patrick B. Ancheta, Diego Sulayman R. Pascua, Resh Vnzi S. Togueño,
Kaye Melina Natividad B. Alamag and Jay Jay F. Manuel

Philippine Science High School – Cordillera Administrative Region Campus, Purok 12, Irisan,
Baguio City, 2600, Philippines

Corresponding Author Email: vashpatrickancheta@gmail.com

Abstract

Machine learning methods are algorithms where machines are not explicitly programmed to do what is tasked but rather, learns how to perform the task. An *m, n, k* game is a game where there is an $m \times n$ grid and two players alternate turns trying to earn k pieces adjacent to each other horizontally, vertically or diagonally. The *m, n, k* game to be used in the research to test the MLMs is Tic-Tac-Toe, configured as 3,3,3. The study utilizes an existing genetic algorithm to be used as a control setup. This genetic algorithm is modified to be controlled by move generators using the Controlled Elite Preservation operator and the resulting genetic algorithms are compared with regard to performance. Using an ANOVA Test at $\alpha = 0.05$, no significant difference in performance was found between the unmodified and modified genetic algorithms. This study provides a backbone for research involved in the transmission of knowledge between "smart" artificial intelligence and "naive" intelligence, raising the question on whether the evolution of a genetic algorithm can be better affected by a move generator in other components.

Keywords: genetic algorithm; machine learning; tic-tac-toe; training methods

Introduction

Machine learning (ML) is vast—it is used in different situations such as spam detectors, web search engines, photo tagging applications and game development (Sharma, 2016). There have been researches that are aimed at improving the implementation of ML in various games. A category of games under investigation through ML is the set of *m, n, k* game games, comprised of games where there is an $m \times n$ grid and two players alternate turns trying to earn k pieces adjacent to each other horizontally, vertically or diagonally (Hayes & Loge, 2016). Among the most common examples of *m, n, k* games are Gō, Othello, and Chess. Tic-Tac-Toe, the game under investigation in this study, is an example of an *m, n, k* game. A Tic-Tac-Toe board is composed of three rows and three columns, and requires three adjacent pieces of the same player to render a win, thus it is considered to have a $3, 3, 3$ configuration.

Improvements in ML have lead to the development of artificial intelligence (AI) players that can beat even the most competitive human players around the world. Machine learning methods (MLMs) are algorithms where machines are not explicitly programmed to do what is tasked. Rather, similar to its namesake, MLM-trained machines are capable of performing tasks given its own internal code without any human interference. In short, the machine *learns* (GeeksforGeeks, n.d.). An example of an MLM is the genetic algorithm (GA).

This study aims to develop multiple GAs with different elite preservation methods and compare their performance in Tic-Tac-Toe based on the possible situations.

This study contributes to the body of knowledge in ML. Through this study, more can be known about how information gained from one method of AI can be passed on to another mechanism of AI through training. This sheds light on how information from one AI player can be transmitted to an MLM such as GA. This, by extension, can improve the comprehension of how machines can learn strategies in games from one with greater skill.

This study focuses only on Tic-Tac-Toe and not other games such as Chess or Gō because it is the simplest game to conduct the research on heuristics, namely the training of the GA under different move generators (MGs). The complexity of the board game is not relevant to the study because the focus of the research is to compare the effectiveness of trained GA organisms given an *m, n, k* game. Applying these heuristics on other *m, n, k* games however are beyond the time frame of the research. Only three GAs were developed in this study. The first is a Python implementation of the GA in the work of Bhatt et al. (2008). Using developed AI, the second and third are modified implementations of the same GA. The performance of each GA is based on how many generations it takes for the GA to find a no-loss first player for Tic-Tac-Toe. This basis for comparison of performance, specifically using the skill of an organism as a first player, is due to the fact that Python is known for being slow. In line with this, indices are 0-based in this paper, as they are in Python.

This study aims to compare the effectiveness of trained genetic algorithm (GA) organisms among each other as applied to Tic-Tac-Toeusing known heuristics into Python code. This will be accomplished by training organisms of an implemented GA using different move generators (MGs) and comparing the development of the performance of trained GA organisms among each other within 500 generations.

Methodology

This Developmental Research is composed of two components: Software Development and Data Collection. In Software Development the code for the move generators and genetic algorithms are initialized, and the efficiency of its implementation is optimized. In Data Collection the genetic algorithms were simulated and after that, the fitness data collected was analyzed using *R*. The independent variable is the elite preservation method. The dependent variable is the number of generations the specific GA takes to find a no-loss solution. Extraneous variables such as the software specifications can be held constant by the use of the same software such as the operating system and the same Python version (3.8.3 64-bit). The study was not affected by hardware specifications as it concerns the number of generations instead of the time taken on the system.

The software was developed and data was collected mainly at the Philippine Science High School – Cordillera Administrative Region Campus. The training and data collection occurred at the aforementioned location on various personal computers. The software was developed with Python 3.8.3 on KDE Neon 5.18 using Visual Studio Code 1.45.1 and hosted on GitHub (https://github.com/MasterToast10/paleo-str-g12). The developmental computer is equipped with 4 GB of Random Access Memory (RAM). Various personal computers were used to perform the training.

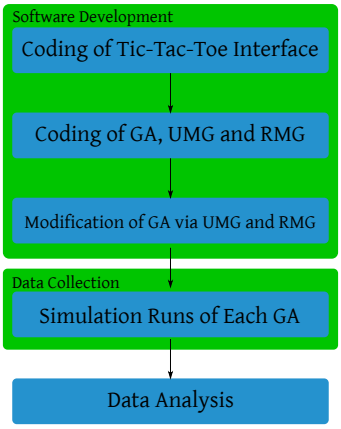

Figure 1. Procedural Flowchart.

The study aims to compare the performance of the genetic algorithms with different elite preservation methods. The three are: an unmodified Python implementation of the work of Bhatt et al. (2008), and two with modified elite preservation methods based on MGs. A preliminary for the accomplishment of such task is a game engine for the interaction between trained GA organisms. This was coded in Python, as application programming interfaces (APIs) for machine learning have already been implemented in Python.

After the development of a platform for play of Tic-Tac-Toe between trained organisms, the development of the required MGs followed. The Random Move Generator (RMG) returns one of all possible moves in Tic-Tac-Toe with equal probability. Another MG is Unbeatable Move Generator (UMG).

UMG is based off of the work of Barrat (2019). It is stated to be unbeatable, however, the Move Generator has a small chance of being beat as the second player. This makes an ideal opponent for trained organisms, given that the GA will find an exploit eventually. In line with this, the GA might be able to use said exploit to have better performance.

The organisms to be trained were implemented in Python as well. The GA was composed of unique genomes with genes for each possible game state with alleles that correspond to the moves to be taken. Following the flow represented in Figure **??**, the operators are adapted from Bhatt et al. (2008). The only aspect of the GA that differs from the implementation of that in Bhatt et al. (2008) is the organism representation.

Organism Representation

Organisms are represented with genomes, as shown in Figure 2. Each gene in the genome represents a game state that considers rotation and symmetry using an implementation of the game-base used by Bhatt et al. (2008). These genes have a corresponding allele that represents the move the organism will take.

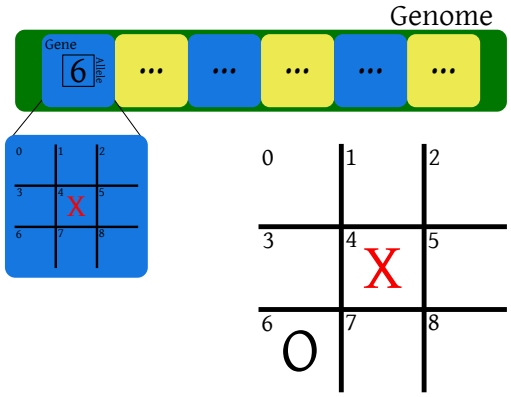

Figure 2. Genome of an organism.

Initial Population

For each game state represented by genes in the genome, the organism is given a random possible move for that game state. Any valid move in a given game state is considered in choosing the allele randomly.

Selection Operator

The following niched fitness equation (Bhatt et al., 2008) was used to modify the implementation of the Stochastic Universal Selection operator by Panchapakesan (2014):

$$f_{\text{niched}}(x_i) = \frac{1}{m} * [1 - f(x_i)]$$

where m is the number of organisms with the same fitness. Using $\frac{1}{m}$ as a factor produces a niching effect (Bhatt et al., 2008). The $[1 - f(x_i)]$ quantity produces the opposite pattern compared to Equation **??**, where instead of zero as the perfect fitness and one as the worst fitness, zero is the worst fitness and one is the perfect fitness. This was done due to the fact that SUS is inherently maximizing (Bhatt et al., 2008). In this study, half of the population is selected via SUS.

Crossover Operator

Similar to that of Bhatt et al. (2008), 50 cross-sites are randomly chosen per parent pair which is used to take alternate series of alleles from each parent. This produces two new offspring with alleles from each parent. The mating pool is linear, with parent pairs formed from a mating pool individual and the individual adjacent to it. If the individual is last in the linear mating pool, it is paired with the first individual in the mating pool.

Mutation Operator

Using random reset mutation, n_m genes are mutated based on the best fitness in the population. This basis is formalized by Bhatt et al. (2008) in the following equation:

$$n_m = 250 * f_{\text{minimum}} + 10$$

Controlled Elite Preservation

In the control genetic algorithm, the organisms from the population before crossover, the population before mutation, and the population after mutation are sorted by fitness in increasing order. The solutions with sorted index $j - 1$ are chosen:

$$j(i) = i + 2N * \frac{(i-1)(i-2)}{(N-1)(N-2)}, \quad i = 1, 2, \ldots, N$$

Modified Controlled Elite Preservation

The following modifications were made to the elite preservation method to produce two other GAs: In UMG-Controlled Elite Preservation, the organisms are sorted by the number of losses of each organism when played against the UMG for 300 games. In RMG-Controlled Elite Preservation, the organisms are sorted by the number of losses of each organism when played against the RMG for 300 games.

In the beginning of the GA the organisms were behaving randomly. The fitness function for the organisms was based on the possible results based on an unpredictable opponent, thus every possible win, loss and draw is computed and plugged into Equation **??** to retrieve the fitness of an organism. Through natural selection the "best" organism prevails. When the "best" organism has zero fitness, the GA stops and snapshots the population for the last time. If no zero-fitness organism is found before 500 generations have passed, the GA stops itself. Using this method, 30 simulations per elite preservation method was run, with 100 organisms per population. The number of generations before a simulation encounters a zero-fitness (based on Equation **??**) or no-loss solution is recorded for each simulation.

$$H_0 : \mu_{\text{UMG}} = \mu_{\text{fitness}} = \mu_{\text{RMG}}$$
$$H_a : \text{The means are not all equal}$$

The data was analyzed through the *R* with the use of Analysis of Variance (ANOVA) which gives the p-value to test the hypotheses at a given confidence interval. The study utilizes $\alpha = 0.10$. Should the alternative hypothesis be true, Tukey Honest Significant Differences (Tukey HSD) is applied to locate the different mean.

Results

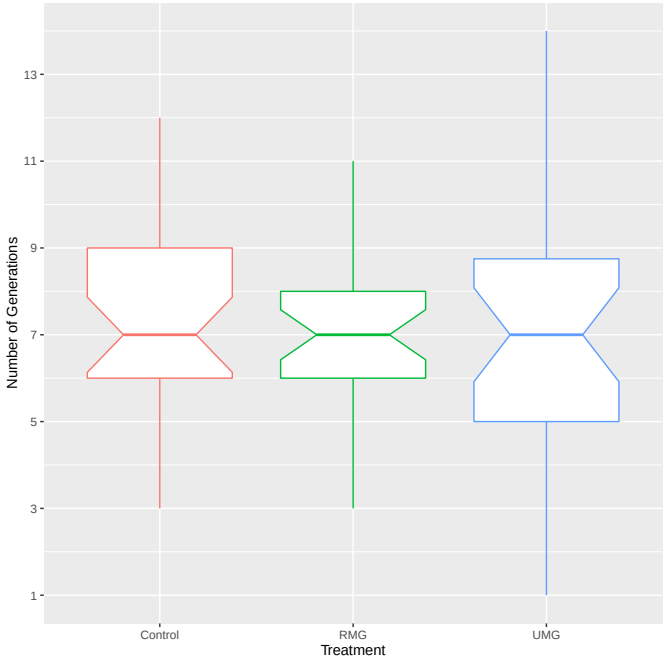

Figure 3. Notched box plot of treatments.

A notched box plot of treatment results is shown in Figure 3. This box plot denotes the interquartile range (IQR) where 50% of the data is located. It is shown that each treatment has a median of seven (7) generations before termination, and the

population median is shown by the notches to be within six to eight generations. The UMG-controlled GA shows a larger spread while the RMG-controlled GA shows more consistent data points.

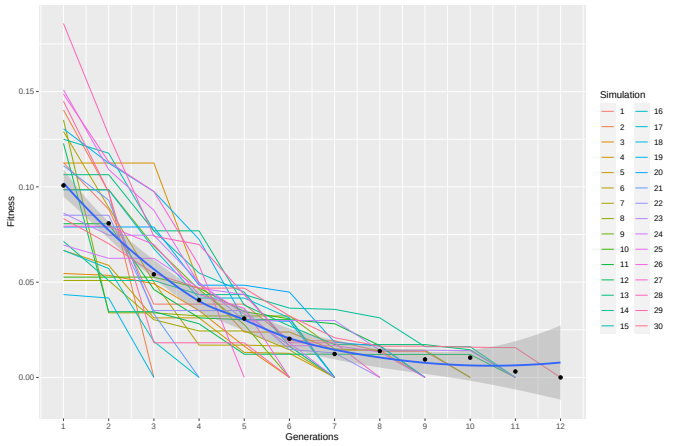

Figure 4. Population-best fitness graph of control.

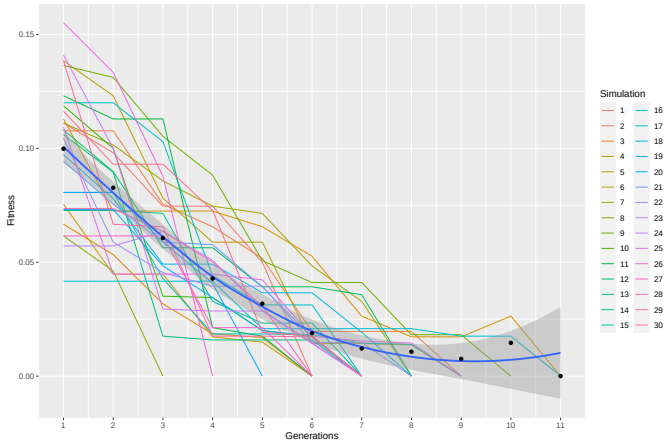

Figure 5. Population-best fitness graph of RMG-controlled.

The large spread in the data points of the UMG-controlled GA is also evident in Figure 6. It is easily shown by the wide standard error bar depicted along the smoothed regression model compared to the standard error bars depicted in Figure 4 and Figure 5. Figure 6 also depicts unique cases where population-best fitness suddenly increases by a large margin.

This evident spread might invalidate the ANOVA test, thus Bartlett's Test for Homogeneity of Variances was applied to the data via *R* (NIST/SEMATECH, 2013). This test resulted in $\chi^2 = 5.1019$, $df = 2$ with a p-value of 0.07801. Said p-value is greater than $\alpha = 0.05$, thus the assumption of homogeneity of variances that is necessary in ANOVA is maintained.

ANOVA was then used to test the null hypothesis that the means of the treatments are equal. This returned an F-value of 0.264 and a p-value of 0.768. The p-value is greater than $\alpha = 0.05$, therefore the null hypothesis that the means of the treatments are equal is not rejected.

It can be noted in Figures 4, 5, and 6 that the GAs stop on or before 15 generations. This is because the GAs can find no-loss solutions before even surpassing the 500 generation mark. The 10-generation worst case tabulated by Bhatt et al. (2008) agrees with these findings. The results for each GA are comparatively lower than the 500-generation upper bound used by Bhatt et al. (2008). This can be explained by the fact that the first player is more likely to win in Tic-Tac-Toe (Cranenburgh et al., 2007), thus the game is inherently biased towards the first player.

The ANOVA test results assert that the modifications have no effect on the number of generations a GA requires to find a no-loss solution. This means that the application of an unbeatable opponent did not promote the genetic evolution of the GA towards

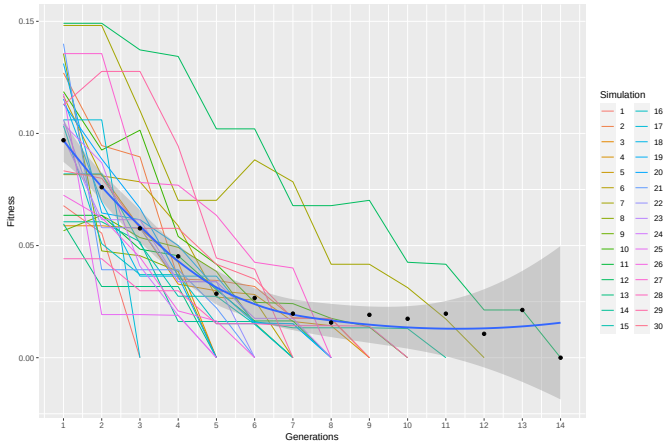

Figure 6. Population-best fitness graph of UMG-controlled.

a no-loss solution. This can imply that either the modification was made in a less-effective component, or the application did not change any behaviour of the GA at all. The latter implication can be disproven by the discrepancies produced by the UMG-controlled GA demonstrated in Figure 6. This behaviour shows that in some cases, the alignments of the UMG goes against the alignments of the fitness function. Such behaviour demonstrates that the UMG modification does in fact, affect the evolution of the GA in no small way.

In the case that the UMG modification was placed in a less-effective component of the GA, it could be implied that there is no effective component in the GA that can accommodate the modification, or that there is one that is not specifically in the Controlled Elite Preservation operator. To prove whether there is an effective component in the GA that can utilize the "unbeatability" of the UMG or not requires extensive research far beyond the scope of this study.

Either way, this study shows that modifying the Controlled Elite Preservation operator of the GA does not significantly improve the performance of the GA. An observation that might be of use, however, is the fact that the UMG-controlled GA shows promise of increase in variability of the GA, which can be used in finding more diverse solutions in the search space.

Summary and Conclusion

In essence, the stated objectives: to train organisms with MGs and to compare their performance, has been achieved through the modifications to the Controlled Elite Preservation operator. This study targeted the Controlled Elite Preservation operator specifically, thus further research is required to acquire more comprehensive data regarding the effects of having a "smart" opponent control the evolution of a GA.

The study resulted in proof that modifying the Controlled Elite Preservation operator does not improve the performance of the GA. This opens up more questions, on whether or not the application of MG will ever affect the behaviour of a GA. It is also recommended to involve testing the performance of the GA as a second player. Furthermore, this research serves as a foundation for more extensive investigations into the effects of "smart" AI on the learning of other AI.

Acknowledgement

We are grateful for our friends and family for their continued support in our continuous lives. We are also thankful to our teachers, research teacher, and research adviser for their unwavering assistance in having the research completed. Without these people, this research would never be successful.

References

Barrat, R. (2019, April 9). *Robbiebarrat/unbeatable_tictactoe*. Retrieved April 27, 2020, from https://github.com/robbiebarrat/unbeatable_tictactoe

Bhatt, A., Varshney, P., & Deb, K. (2008, July 12). In search of no-loss strategies for the game of tic-tac-toe using a customized genetic algorithm. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. https://doi.org/10.1145/1389095.1389269

Cranenburgh, A. V., Samid, R., & van Someran, M. (2007). Tic-Tac-Toe.

GeeksforGeeks. (n.d.). *Machine learning*. Retrieved October 22, 2019, from https://www.geeksforgeeks.org/machine-learning/

Hayes, N., & Loge, T. (2016). Developing a memory efficient algorithm for playing m, n, k games. https://www.micsymposium.org/mics2016/Papers/MICS_2016_paper_28.pdf

NIST/SEMATECH. (2013, October 30). *1.3.5.7. Bartlett's Test*. Retrieved May 19, 2020, from https://www.itl.nist.gov/div898/handbook/eda/section3/eda357.htm

Panchapakesan, A. (2014, March 30). *Stochastic universal sampling GA in python*. Retrieved May 12, 2020, from https://stackoverflow.com/a/22750088

Sharma, A. (2016, January 11). *Machine learning - Applications*. Retrieved October 22, 2019, from https://www.geeksforgeeks.org/machine-learning-introduction/