Ludo - Genetic Algorithm

Sina Pour Soltani
siesm17@student.sdu.dk

University of Southern Denmark

Abstract. In this paper a Genetic Algorithm (GA) is used to teach an AI agent to play Ludo. Two different methods are explored and compared on the basis of the GA; one where the GA learns the weights of a multilayered perceptron (MLP), and another where the GA evolves to arrive at the (locally) optimal weights in a State Transition Function (STF). Results show that the population in the GA arrives at a higher average *win-rate* of 75.3% with the STF compared to a win-rate of 29.0% using the MLP.

1 Introduction

Ludo is a game of chance and calculated strategy. In this paper, multiple methods and different state representations are explored, with the objective of increasing the winning chances of an AI-agent at Ludo. The main algorithm used to increase the win-rate of the agent is the Genetic Algorithm. The algorithm is tried and tested in two separate but comparable ways.

In the first method; GA is used to learn the optimal weights of a multilayer perceptron with nodes in hidden layers, based on the die and the positional information of the pieces on the board.

In the second method; a formal and efficient, albeit reduced, representation is derived to give information of the state of the Ludo board, which the GA utilises to determine which of its pieces are more favourable to move. GA is then used to arrive at a set of weights that determine which state transition is advantageous based on the information in the reduced state representation.

This paper is structured as follows: Section 2 describes the concepts and details of the implemented GA that both methods rely upon. Hereafter, each of the methods and their respective differences are illuminated. Section 3 provides the results of the tested methods. Section 4 discusses the results before concluding on the findings in Section 5.

The algorithm is implemented in Python using two notable packages; LUDOpy [1] to simulate the Ludo game and PyTorch [2] for implementing the architecture of the MLP.

2 Methods

The Genetic Algorithm is based on a few key concepts; fitness evaluation, selection and reproduction. A Population of Individuals cycle through this process, ultimately optimising on the given problem defined by the fitness function. [3]

In this implementation, the chromosome of the Individuals was bit-encoded and could be translated into weights for the model when needed. The number of bits in a weight is an adjustable parameter, which is used to control the number of levels and the resolution of the weights (see figure 1).

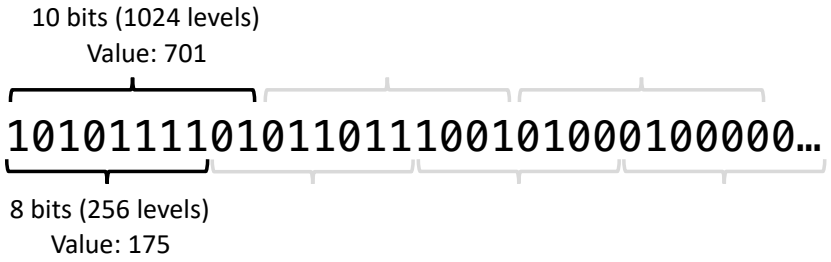

Fig. 1. The chromosome of the Individuals in the Population.

The fitness of an individual was calculated based on the number of wins in the Ludo game, as that is ultimately what the program should optimise for. In the implementation, each Individual plays 30 games before their respective fitness is assessed. Initial testing in multiple configurations on a simpler problem (fitness evaluated based on the number of 0's in the chromosome) revealed the values for the different parameters and methods, which solved the problem the fastest.

A standard single-population GA with 100 members, two-point crossover with a 65% crossover rate, bit flip mutation with a rate of 0.1%, with ranked selection and generational replacement seemed to optimise the problem the fastest and most efficiently. This configuration will set the basis for the tests performed in this paper. When needed the bit-encoded chromosome of the Individual was translated to a vector of integers followed by a normalisation between to end at weights in the range [-0.5, 0.5]. [4]

In the following, the two methods for increasing the win-rate with GA will be described.

2.1 Multilayer Perceptron

The idea behind this method is to naively use the information embedded in the positional observation of all the pieces and the die. Since this information describes the positions of the pieces as well as their relative distances, all the information needed to perform a qualified move is present. All other relevant information is constant and can thereby be learned. This information is forwarded through the fully connected layers of a multilayered perceptron with the aim of choosing a valid piece for moving (see figure 2).

All of the weights of the MLP are embedded in the chromosome of an Individual. Using the GA, the objective is to arrive at the optimal set of weights that, given the state of the board, chooses the optimal piece to move.

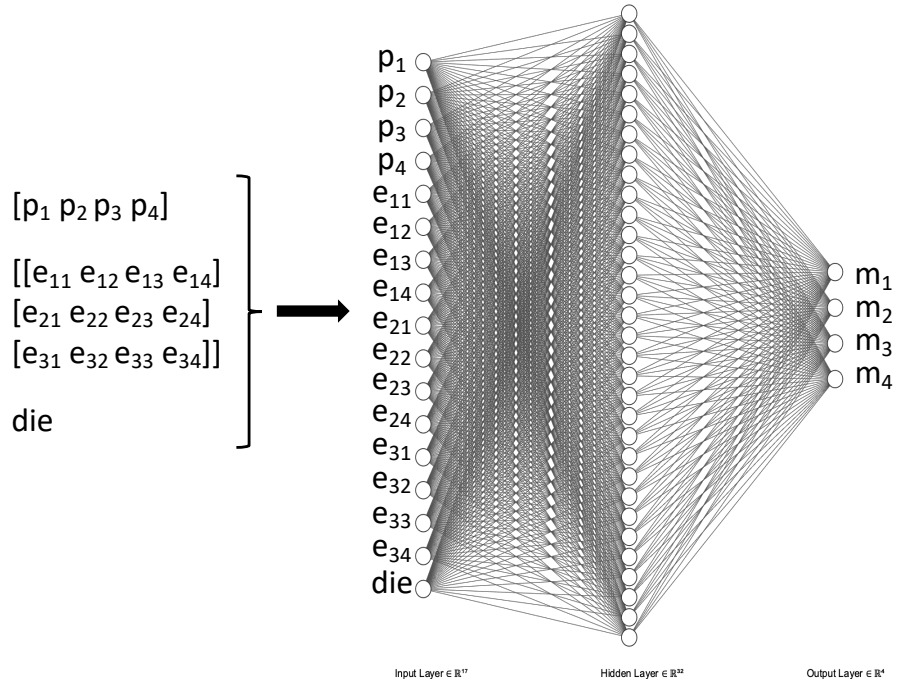

Fig. 2. The structure of the multilayer perceptron with the embedding of the die and the positions of the pieces of the player and enemies.

The MLP is designed with an input dimension of 17 accommodating the positions of the pieces of the player and the enemies as well as the die. The MLP is also comprised of a hidden layer with 32 nodes, which are connected to the output layer with 4 nodes; indicating which of the pieces to move. The ReLU activation function is used between the layers, and a softmax function is used on the output to normalise the output to a probability distribution. Finally, the node with the highest probability corresponding to its piece is chosen to move. If the MLP arrives at an unmovable piece when possible moves exist, the fitness score is reduced and one of the possible moves are chosen by random.

2.2 State Transition Function

In the State Transition Function, a reduced representation of the Ludo state is utilised. Here the state information of the pieces in the Ludo game are lumped into nine distinct binary descriptors. This vector of state descriptors describes the state of the Ludo board after one of the pieces has been chosen to move. As there are at most four pieces to move, there will also be the same number of

State Transition Vectors (STV). Some of the descriptors in the State Transition Vector only relate to the moving piece, where the rest can be applied to both moving and stationary pieces. The descriptors of the State Transition Vector can be seen listed below:

1. Move out: The moving piece can exit home
2. Travel: The moving piece can land on a star and travel to next star
3. Hit: The moving piece can hit an enemy player home
4. Suicide: The moving piece can hit itself home

---

5. Safe: The piece is on a square where it can't be hit home
6. Vulnerable: The piece is 1-6 fields ahead of enemy and not safe
7. Attacking: The piece is 1-6 behind enemy
8. Runway: The piece is on the the 6 squares before goal
9. Finish: The piece is on the goal and out of the game

When it is the agent's turn, the state of all the pieces are evaluated. Alternately, the state is evaluated for each movable piece, as well as the remaining stationary pieces. These states are then added together to arrive at a single State Transition Vector, which describes the state of the board, depending on which of the pieces was chosen to move (see figure 3).

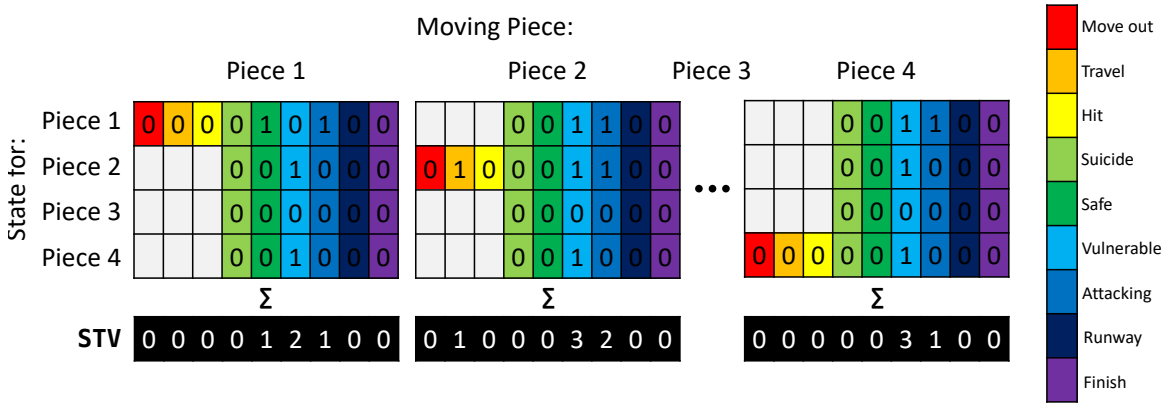

Fig. 3. Visualisation of the binary descriptors that make out the states of the individual pieces, and how these are summed to the final State Transition Vector.

The dot product between the STV and the weights of an Individual is then calculated to arrive at a score for each of the STV. In a sense, the Individual operates as a single perceptron, that outputs a dot product between the input vector (STV) and the weights of the perceptron (chromosome).

The STV with the highest score indicates the best possible move to take, and the corresponding piece is moved. Therefore, the weights of the Individual in essence represent the preference for the different state descriptors. By weighing preferable states positively and undesirable states negatively, the agent picks the STV, that it deems most favourable.

3 Results

In this section the results of running the Genetic Algorithm for 100 generations are displayed. Figures 4 and 5 show boxplots of the progression in wins of the Individuals throughout the evolution of the Population. Figure 3 shows the comparison between the win-rates of the MLP-method and the STF-method.

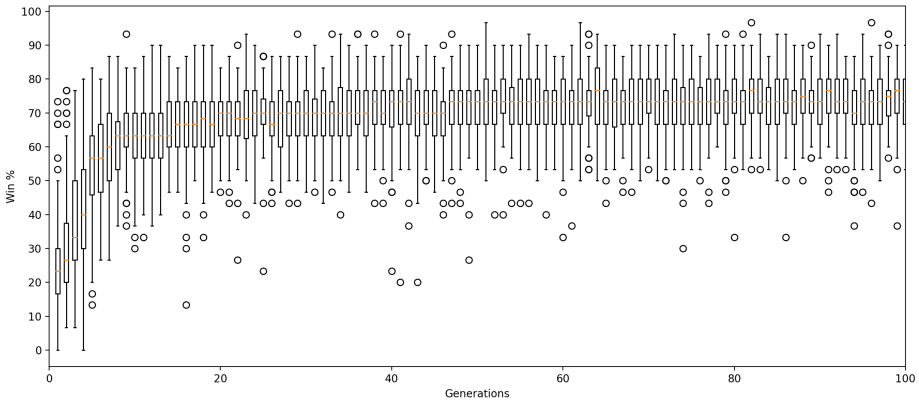

Fig. 4. Wins of the population in the STF.

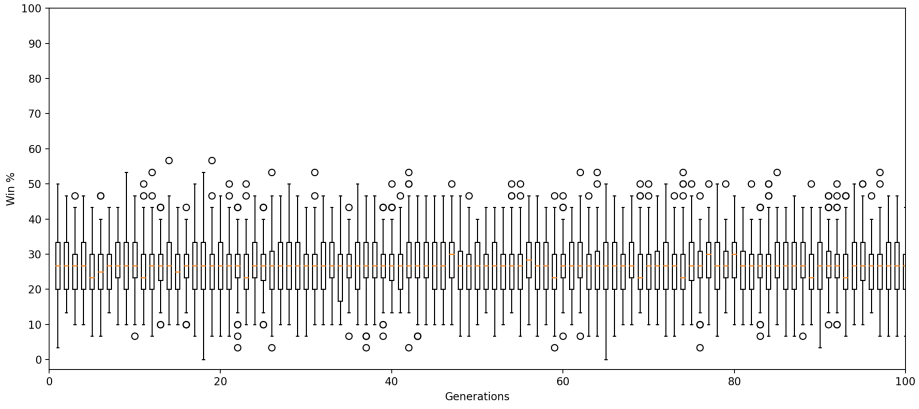

Fig. 5. Wins of the population in the MLP.

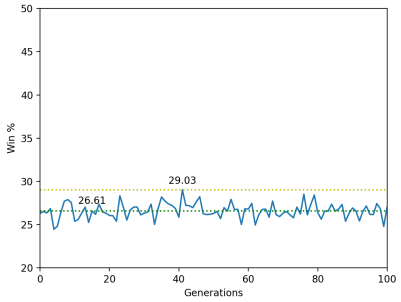

Fig. 6. Win-rate of the MLP.

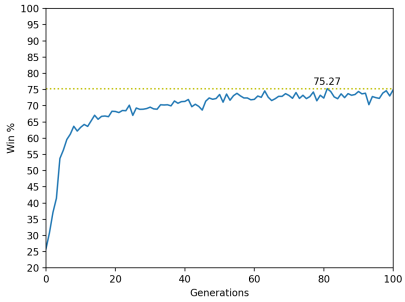

Fig. 7. Win-rate of the STF.

4 Discussion

It is apparent from figure 5 that the method using the MLP to increase the win-rate of the GA agent does not improve in any substantial way. From figure 6 it is clear to see that the win-rate remains approximately at the starting level. From a completely random Ludo player, one would expect an average win-rate of 25%, which experiments confirm. The MLP however, manages to eek just slightly above this baseline with an average win-rate of all generations of 26.6%, with the best generation reaching an average win-rate of 29.0%. This indicates that the MLP is able to make *some* reasonable decisions, but never gets close to an optimal solution in the search space.

Meanwhile, the figures 4 and 7 clearly show that the method using the State Transition Function is very successful in improving the win-rate of the agent substantially. This method achieved the highest mean win-rate of a Population of 75.3%. Furthermore, many Individuals within the interquartile range achieve win-rates above 80 and early 90's, with outliers even above 95%.

Before seeing the results, a fair assumption could be that the MLP would outperform the STF since it has full state information. From this information, it would be able to select pieces, which have already been invested many moves in i.e. pieces that have gotten far on the board. This is something that is not possible with the reduced state representation, as it only shows a relative state.

A possible explanation for the big discrepancy between the two methods could be the difference in lengths of their respective chromosomes. With a bit-resolution of 10, the length of the chromosomes for the Individuals in the MLP-method is 7080, while it is only 90 for the Individuals in the STF-method. As it is the relationship between the weights, that are the most important factor in arriving at a solution in the two methods, there is a higher probability of convergence with fewer weights i.e. a shorter chromosome.

The GA was also only run on the configuration that was reached by a simpler problem. It could be that the relatively high crossover rate and the low mutation

rate were optimal for the simpler problem but completely wrong for a system with this many bits in its chromosome.

It is notable that only one configuration of the MLP was tried. It could be that the architecture tried in this paper was not able to converge within 100 generations in its huge search space. However, it could be that a different structure or encoding of the state would perform better. A different number of hidden layers and number of nodes could be examined. Furthermore, it could be that an optimal set of weights would be easier to reach with a one-hot encoding of the input vector. These different approaches have not been explored in this paper.

Another thing that would be interesting to examine, would be to train the weights of the MLP with a more suitable algorithm such as backpropagation while using the moves from the STF as targets in a setting with supervised learning.

5 Conclusion

In this paper, the Genetic Algorithm was explored along with two different methods with the objective of increasing the win-rate of a GA agent playing Ludo.

One method naively applied the full state information from the Ludo game, and forwarded this through a multilayer perceptron; the weights of which were determined by the chromosome of an Individual in a Population. The output of the MLP decides which of the four pieces to move.

The second method used a reduced state representation with broader and relative binary state descriptors. These descriptors gave an indication of the state of each piece on the Ludo board. Adding the states of a moving piece alongside the remaining stationary pieces produced a State Transition Vector. The dot product between the State Transition Vector and the weights in the chromosome of an Individual was used to decide which piece to move.

Running the Genetic Algorithm for 100 generations provided results for the two different methods. The results were clear in showing that the reduced state representation with the State Transition Vector produced a more competent agent with the highest average win-rate of a Population being 75.3%. In comparison, the Population with the average win-rate for the MLP-method scored 29.0%.

6 Acknowledgements

Many stimulating discussions were made with Christian Jannick Eberhardt, Jakob Grøftehauge Rasmussen and Jens Otto Hee Iversen, which inspired the work in this paper. Furthermore, Jakob read the paper and gave feedback. I am grateful for all of them.

References

1. SimonLBSoerensen. (2021, mar) Ludopy. [Online]. Available: https://github.com/SimonLBSoerensen/LUDOpy
2. [Online]. Available: https://pytorch.org/
3. P. Manoonpong, "Evolutionary robotics," *Tools of Artificial Intelligence*, 2021.
4. SinaPourSoltani. (2021, may) Ludoga. [Online]. Available: https://github.com/SinaPourSoltani/LudoGA