5 x 5 x 5 Tic-Tac-Toe AI Implemented by Genetic Algorithm

Alex Ferraco, Yuchen Wang, Christopher Todd.

Department of Computer Science
College of Engineering, Pennsylvania State University
State College, U.S.A.

Abstract—This paper explores developing a genetic algorithm to play a 5 by 5 by 5 tic-tac-toe game (also known as naughts-and-crosses).

INTRODUCTION

A normal game of 3 by 3 tic-tac-toe, hereafter referred to as “simple TTT”, is a very simple game that can be easily mastered by an adult after just a few games played. Though this game is simple, and can be mastered by children, it takes a considerable amount of knowledge and effort to program an AI to play a perfect game of simple TTT. In simple TTT there are 9! Possible games that can be played (assuming that a game doesn’t end when 3 in a row is made). This presents a very large search space, and it makes for an impractical brute force search. For a 5 by 5 by 5 game of tic-tac-toe, hereafter referred to as “complex TTT”, the search space would be approximately 125! (Clearly not fit for a brute force search). For our solution we will be using a genetic algorithm, this will be discussed in detail later in the paper.

EXISTING RESEARCH

Much of the existing research around using genetic algorithms to solve TTT like problems is limited to simple TTT. In Bhatt, Varshney, and Deb [2], the authors are in search of a no loss strategy for simple TTT that uses a genetic algorithm. They are able to achieve this by reducing the number of board states by rotating and flipping the board. As the title of the article suggests, the goal was to evolve a no loss strategy, and thus, the fitness of each chromosome is calculated by “number of games lost” divided by “number of games played”, here, lower fitness scores are good, and a fitness score of zero is “perfect”. This will do perfectly well to evolve a no loss algorithm, but it will take a slightly more complex fitness function to evolve a solution that optimizes a winning strategy.

Another idea we were able to find in our research was how to essentially “evolve” the chromosomes. Morelli [1] describes a process he used that pitted chromosomes against each other. This idea really helped us in our development of a GA algorithm. One of the most important parts of a GA is being able to define a fitness function. We modeled our fitness function after this idea, in that our chromosomes play games against each other to obtain a fitness number. This is discussed in more detail in the actual description of our method below.

We also found some relevant information regarding the population size and number of generations to run the algorithm. We had discussed the information we found in Gathercole and Ross [3] and thoroughly enjoyed their idea. They go on to show how smaller populations over many generations can perform better than large populations over small generations. This narrowed down our decision of how to select these properties; however we cannot conclude an exact quantity because it will fluctuate when we begin testing.

A problem we came across in developing our GA was how to encode our strategy into a chromosome. We went through several ineffective and inefficient methods before we came across a concept that we were able to piggy back off of. The idea that a chromosome is a strategy with a set of rules came up in Sutton and Baito [5] and Hochmuth [4]. This is where we started to think about configuring a chromosome into a set of rules. We essentially applied the idea of natural selection to a large amount of rules to figure out which ones are the best. This was a major breakthrough for us and it brought together all of the fragmented ideas we had come up with so far. At this point we had discovered plenty of workable ideas and theory’s and we began with the design of our methodology.

THEORETICAL IMPLEMENTATION

Defining the problem space:

The first step in any genetic algorithm design is to determine the problem and find a way to represent it. For a game of tic-tac-toe, the obvious goal is to win. A strategy must be developed to achieve this goal, and it must be able to be encoded in a chromosome. A strategy is described as follows:

- A strategy is a chromosome and it will be a list of rules that the player will follow throughout a game
- There will be a large set of predefined rules to select from
- Each gene in a chromosome will encode one rule
Each rule will be either turned on or turned off
A chromosome will be a collection of genes
The player will only use the rules activated by the chromosome

By using this model for our strategy, we will encode it into a chromosome. A chromosome will be a list of genes that are 1 bit each. Each chromosome will have the same amount of genes as there are rules. The way to determine whether or not a rule will be active depends on the value of each gene. If the gene is a 1, then the rule corresponding to that gene index in the chromosome will be active. If the gene is a 0, the rule will be inactive.

A chromosome's fitness will be evaluated based on how often the chromosome wins. Each chromosome will play an undetermined amount of games against other chromosomes. The fitness will be calculated based on a point system for each win, tie, and loss. The points will be decided through experimentation and testing, but a win would be worth more points than a tie, and a tie would be worth more points than a loss.

Since the genes are each 1 bit, the standard crossover and mutation operators can be applied to the chromosome. Again, the rates of each operator will be dependent on testing, but our research shows that the crossover rate is usually .3 and the mutation rate is between .01 and .001.

**EVOLUTION STRATEGY**

The Theory:

Since every chromosome is a strategy, the genetic algorithm should be able to cross and mutate a strategy into an optimal strategy that would yield the highest winning chance.

The Process:

First, an initial population of chromosomes will be created randomly. Each chromosome in the population will play against all of the other chromosomes. The amount of times they will play should be an even number, so that they can alternate moving first. We will keep track of the amount of wins, ties, and losses of each chromosome. The overall record will be converted to a number that is calculated by assigning different point values to wins, ties, and losses and sums them. This number will be the fitness number for the given chromosome. The selection for reproduction will use a roulette system. This system will select chromosomes according to how high their fitness is. The higher the fitness, the better the chance it has to be selected. Every chromosome pair selected will undergo crossover and mutation based on the respective operator rates and produce the next generation. Then, the new generation will obtain it’s fitness values in the same fashion and this is to be compared against their parents. The chromosome with higher value will be kept in the population and the lower valued chromosomes will be eliminated. This process repeats until the new generation has the same population as the initial population. We will generate some number of generations that will be fine-tuned later on. In the final and terminating generation, the chromosome with the highest fitness number will be our optimal strategy and our solution. The program will play the game in class based off of this chromosome.

**REFERENCES**


