Using Genetic Algorithm to Solve Game of Go-Moku

Sanjay M Shah
SGVU, Jaipur

Dharm Singh
College of Technology and Engineering, MPUAT
Udaipur, India

J.S Shah
L D College of Engineering, Ahmedabad

ABSTRACT

Genetic algorithm is a stochastic parallel beam search that can be applied to many typical search problems. This paper describes a genetic algorithmic approach to a problem in artificial intelligence. During the process of evolution, the environment cooperates with the population by continuously making itself friendlier so as to lower the evolutionary pressure. Evaluations show the performance of this approach seems considerably effective in solving this type of board games. Game-playing programs are often described as being a combination of search and knowledge. Board Games provide dynamic environments that make them ideal area of computational intelligence theories, architectures, and algorithms. Evolutionary algorithms such as Genetic algorithm are applied to the game playing because of the very large state space of the problem. This paper mainly highlights how genetic algorithm can be applied to game of Go-moku.

General Terms

Go-Moku, Genetic Algorithm

Keywords

Population, Chromosome, Fitness Function, Genetic operators.

1. INTRODUCTION

Game playing domain has dominated as one of the extensively studied areas of artificial intelligence. Almost all board games require sophisticated intelligence in a well-defined problem where success is easily measured. Most research in game playing has centered on creating deeper searches through the possible game scenarios.

Through games, efficiency of AI working can be measured in terms of capability to acquire intelligence without putting human lives or property at risk. The old techniques of artificial intelligence work well with games, and to a large extent, such techniques were developed, tested and improvised for such games. [1][2]

It is very much evident that the application domain of game playing programs is primarily focused on optimization problem. The game playing programs have two major considerations:

- Complexity of decision
- Space Complexity i.e. the search space.

These games provide challenges in the form of guiding the evolution with the use of human knowledge and achieving successful and intelligent game playing behavior [3][4]

1.1 Introduction of Game

This is a board game normally played on 15*15 or 19*19 size square board. Go-moku is a two-person, zero-sum deterministic finite board game with perfect information, originated in Japan. It is also known by a name Five-in-line. It is a specific form of a general game connect-X, where X=5 for Go-moku game. Two-person zero-sum games are characterized by the fact that either one player wins and the other loses, or neither player wins resulting in a draw or tie. It is not possible for both players to win. Deterministic games are games where there are no random events based on event of chance or luck such as the roll of a dice. So, there is no blind move. In addition to that Go-moku is a finite game, because there are a finite number of moves. Go-moku has perfect information, because all the game information such as piece position is available to all players. [5]

It has very simple rules but a very complex game. The two players alternate their moves. The move can be made in any blank position in the board. The player who places 5 pieces in sequence wins the game. The sequence can be horizontal, vertical or diagonal. The player having black piece starts the first move.

1.2 Basic definitions

Threat: A threat is a move to which you react. The most simple threat is four. It means series of four same colored pieces. If the opponent does not react in the next move, then the player owning the threat will win in the next move. Figure 1[6] shows various threats in the game.

- Four –a
- Open Four –b
- Three –c
- Open Three –d

Threat category: With each threat we can associate a category, which is nothing but the number of undefended moves required to win. For example, the threat four has class 1 and threat three has class 2 etc.

Threat sequence: It is a series of single attacker moves and one or more defending moves. Naturally, last attacking move reaches a goal state.

1.3 Game Complexity

In Go-moku for 19*19 board, there are total 361 board locations. Each location can have any of three positions, white, black or empty. Thus total state space search is 3361.

2. RELATED WORK

In general, there is very limited number of studies on applying genetic algorithm to the game of Go-Moku. L. Victor Allis, in his Ph D thesis titled, “Searching for Solutions in Games and Artificial Intelligence”, applied db-search and pn-search methods on Go-Moku. But the search was not optimal and was not selecting the shortest path to the goal state.

Marco Kunze and Sebastian Nowozin in their study “An AI for Gomoku/Wuziqi − and more...” also tried to use alpha-beta cut-off procedure in order to reduce the search area[7].

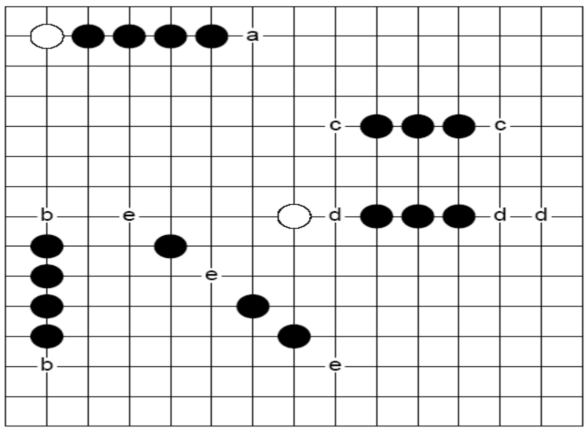

Fig-1 Threats in Go-Moku

3. APPROACH

I have tried to develop a system that consists of two parts: a GUI module and a GA module. The GUI module builds up an environment which consists of a single 19*19 board. The GA module produces a population that starts evolving based on the generated environment.

3.1 Representation of the board and moving strategy

It provides a simulated board of 19*19 size. The board configuration shows the state of the game, and reflects the changes on the board after a specific move by either player.

As it is a two player game, there can not be a fixed moving strategy. The next move will be dependent on the threats generated by the opponent and the threat category.As the player makes a move, the corresponding piece is shown at the appropriate position on the board.

3.2 Genetic Algorithm

From an evolutionary perspective, we can view the Go-Moku board created from the previous module as the environment in which players play their move to reach the goal state of the game i.e connect-5.

Figure-2 explains the general genetic algorithm with its main operations and its repetitions. In practice, Genetic algorithms have been applied to a broad range of learning and optimization problems through a set of Genetic Parameters shown in figure 1 which depicts the cycle of Genetic process. [8]

3.2.1 Individuals and Population:

Each individual chromosome is represented with three different values, x, y and fitness value. The x and y values show the board position and fitness value show how fit the move will be in terms of fitness function value. This is to be carried out for each free position on the board. There are two categories of free position on the board, such as free position in neighboring zone and free position elsewhere in the board. Each element on the board can take any of the four values[10].

- 1= Free position in neighboring zone,
- 0 =Free position,
- 1= Computer player (using GA) piece,
- 2= Human player piece.

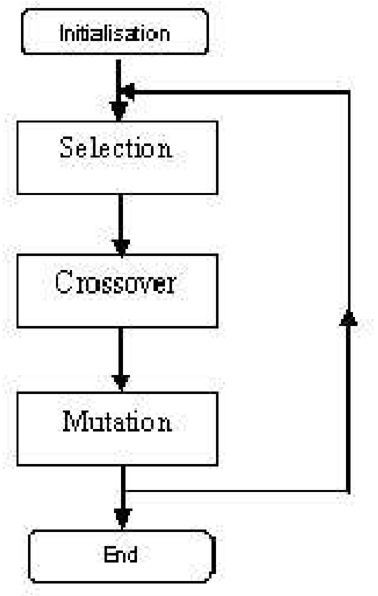

Fig-2 Genetic parameter cycle

The neighboring zone of considered board position is the set of all occupied positions i.e. positions with value 1 or 2. Population size of 150 is maintained throughout the process. Higher the population size, higher the diversity in the population, which in turn results into better outcome. But a very high size of population tends to make system slow and thus inefficient in terms of response. Each time, a new generation is created from the previous one by performing three main operators of genetic algorithm namely in following sequence - copying, mutating, and crossing-over operations.

3.2.2 Crossover and mutation

It is hard to define crossover than mutation. Each chromosomal part from a parent represents a local optimum; however, two local optimums rarely combine to create a global optimum. On the contrary, it is even possible that such a combination will result in a relatively less effective strategy. Rate of cross over selected is 0.5.

3.2.3 Fitness Function

Fitness function value guides the genetic algorithm search optimization process. As the genetic algorithm evaluation process proceeds, the fitness value of each board position is calculated as per the fitness function.

Our proposed fitness function which uses following three measurements to rate a move:

- The possibilities the move gives us to build rows of five stones in the future. That means, if no other criteria work, we will set the stone that we, we can build most possible combinations of rows of five stones over it.
- Building up long lines (or combinations of long lines) to work towards threats and to prefer special kinds of threats.
- The identification of a winning situation.
- The general form of fitness function for board game is like:

$$f(s) = w1* f1 + w2*f2 +w3 * f3 + ...$$

Where, f1, f2, f3 etc are usually based on human strategic knowledge of the game in terms of threats and opportunities in the game, while w1,w2, w3 etc are weightage associated with each f in the formula. Considering above, we propose following fitness function.

f(s) = 4800 * (number of four structures in neighborhood)

+ 97 * (number of three structures in neighborhood)

+ 17 * (number of two structures in neighborhood)

In above fitness function, the four structures is given more weight age in comparison with three structure and so on. We try to maximize the fitness function.

At every generation, fitness value of each chromosome is calculated using fitness function. If fitness of two chromosomes is equal, then the mutation rate is increased, in order to help the genetic evolution get out of issues like local maxima. Once there is an improvement in the overall fitness, the original mutation rate is restored to continue evolution as normal. If the search does not improve fitness value, then new set of initial population is generated using new randomly generated seed value. [9]

4. EXPERIMENT AND RESULTS

The program is implemented in C, on a AMD Phenom II X3 720, 2.81 GHz processor with 2GB RAM, windows vista OS. We are mainly interested in finding how effective the algorithm is within reasonable search time. In addition, we want to know how efficient the algorithm is in terms of time used in making move by computer.

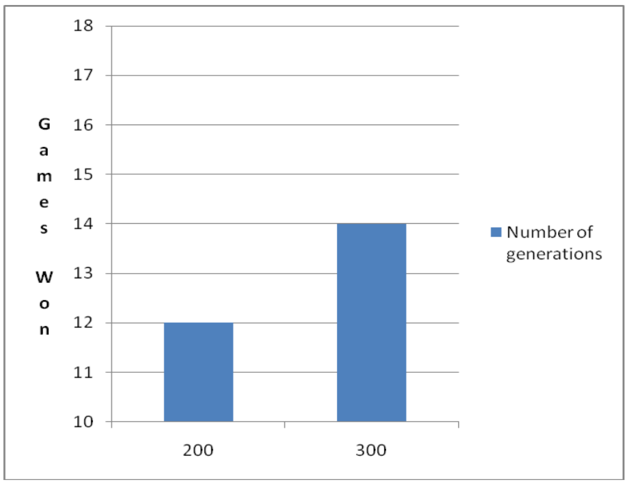

Fig-3 Comparison of games won for 200 and 300 generations

Among 18 testing case games, the algorithm has an encouraging winning rate of 66.7% when 200 generations is allowed maximally. An even better result is observed in the 300-generation group where 77.7% of the games are won by the algorithm. More specifically, the graph in Fig-3 explains the number of won games in each category out of the 18 test games carried out.

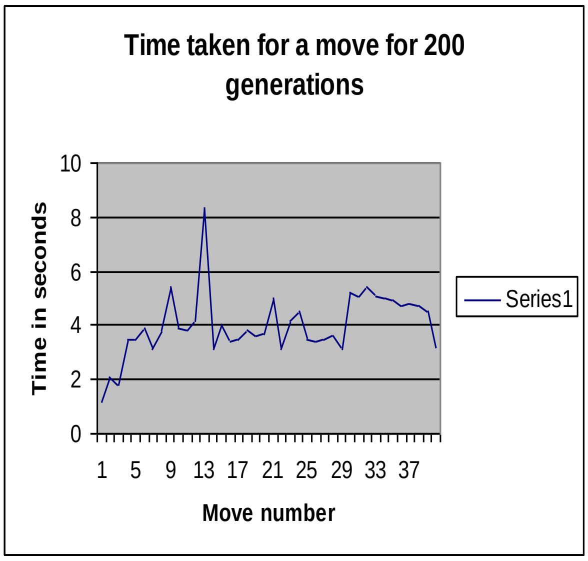

Fig -4 Time for making a move for 200 generations

Considering the trade-off between time and effectiveness, we consider it is an acceptable result. The case with 300 generations shows improvement in terms of % won but it consumes more time in making the computer player move. By performing statistical analysis on the data, we found that there exists a significant relationship between larger generation parameters and higher successful rate.

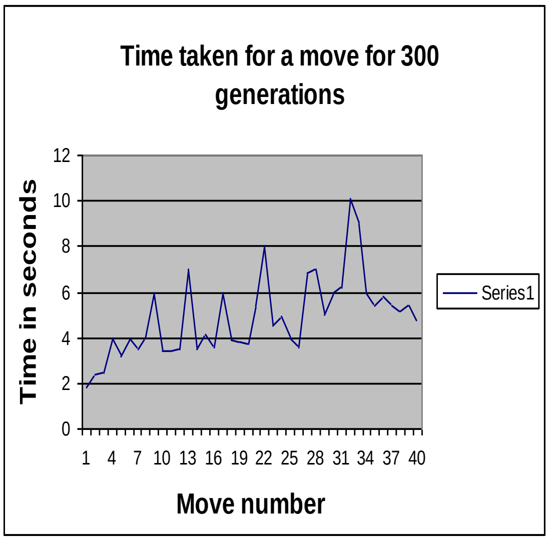

Fig -5 Time for making a move for 300 generations

The overall efficiency of the genetic algorithm is rather satisfactory. In case of 200 generations, the average time used in deciding the move from computer is 4 seconds, with the fastest case taking 1.2 seconds and the slowest 8.3 seconds. Fig 4 shows for 40 sample moves and how much time the algorithm needs before finding a move.In case of 300 generations, the average time consumed is 4.88 seconds, with the fastest case taking 1.8 seconds and the slowest 10.1 seconds. Fig 5 shows how much time the algorithm needs before finding a move.From figure 4 and 5, it is very clear that, though 300 generations of GA gives more success rate but at the cost of extra time. Even in the cases of initial stages, more time taken is noticeable.

5. CONCLUSION

In this paper, I have described an approach to solve the game of Go-Moku using a genetic approach, how exactly the genetic parameter – generations affects the performance in terms of % won game and the average time taken in deciding a move. The resulting system not only solves the problem but also with a considerable efficiency also. Further success can be achieved by tuning the genetic parameters and the fitness function.

6. REFERENCES

[1] Hong, J.-H. and Cho, S.-B. (2004). Evolution of emergent behaviors for shooting game characters in robocode. In Evolutionary Computation, 2004. CEC2004. Congress on Evolutionary Computation, volume 1, pages 634–638, Piscataway, NJ. IEEE.

[2] J. Clune. Heuristic evaluation functions for general game playing. In Proc. of AAAI, 1134–1139, 2007.

[3] Matt Gilgenbach. Fun game AI design for beginners. In Steve Rabin, editor, AI Game Programming Wisdom 3, 2006.

[4] S. Schiffel and M. Thielscher. A multiagent semantics for the game description language. In Proc. of the Int.'l Conf. on Agents and Artificial Intelligence, Porto 2009. Springer LNCS.

[5] T. Srinivasan, P.J.S. Srikanth, K. Praveen and L. Harish Subramaniam, "AI Game Playing Approach for Fast Processor Allocation in Hypercube Systems using Veitch diagram (AIPA)", IADIS International Conference on Applied Computing 2005, vol. 1, Feb. 2005, pp. 65-72.

[6] L. Victor Allis, "Searching for solutions in Games and Artificial Intelligence", Ph D Thesis

[7] Marco Kunze and Sebastian Nowozin in their study "An AI for Gomoku/Wuziqi − and more..."

[8] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and and Machine Learning. Reading, MA: Addison-Wesley.

[9] Ting Qian "Using Genetic Algorithm to Solve Sliding Tile Puzzles".

[10] Sanjay M Shah, Dharm Singh, Chirag S Thaker "Optimization of Fitness Function through Evolutionary Game Learning", Evolution in Networks and Computer Communications-2011, A Special Issue from IJCA.