Optimization of Fitness Function through Evolutionary Game Learning

Sanjay M Shah
Faculty of Engineering
Suresh Gyan Vihar University
Jaipur, India

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

Chirag S Thaker
Faculty of Engineering
Suresh Gyan Vihar University
Jaipur, India

ABSTRACT

Game playing has been one of the main areas of application of Artificial intelligence and programs are often described as being a combination of search and knowledge. The Board Games are very popular due to their nature provide dynamic environments that make them ideal area of computational intelligence theories, architectures, and algorithms. For almost all the board games building a quality evaluation function is usually a challenging work and requires lot of manual hard work and luck. The quality of the evaluation function is determined by its accuracy, relevance, cost and outcome. Good evaluation function must address all these parameters and then the weighed results are added to an evaluation function experimentally.

Almost all board games have very large state space. Due to this nature of board games, evolutionary algorithms such as Genetic algorithm are applied to the game playing. In natural evolution, the fitness of an individual is defined with respect to its competitors and collaborators, as well as to the environment. Evolutionary algorithms follow the same path to evolve game playing programs. Go-moku (Five-in-Line), the board game, is a variant of a Game of GO. This paper mainly highlights application of genetic algorithm to Go-moku and using genetic operators tries to find out fitness values through linear evaluation function applying genetic operators through linear evaluation function.

General Terms

Deterministic Games, Board games, Go-Moku, Genetic Parameters, Chromosome, Fitness function **et. al.**

Keywords

Open four, split three , game learning

1. INTRODUCTION

Game playing is one of the oldest and most extensively studied areas of artificial intelligence. Sophisticated intelligence is required in a well-defined problem where success is easily measured. Games have therefore proven to be important domains for studying problem solving techniques. Most of the research in game playing has centered on creating efficient deeper searches through the possible game scenarios. Games are one of the means to measure efficiency of AI algorithm 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].

This paper presents an approach to game playing by evolving artificial game-playing by taking genetic approach.

In 21st century, the easy and affordable availability of very fast hardware and software tools has changed the field of programming drastically. This has made it possible to simulate complex physical learning environments, resulting in an exploration of artificially improved soft cognitive moves by computer programs in all sorts of board games. Game playing programs have become a facet of many people's routine lives [4].

Due to very high state complexity of almost all traditional board games, it has become AI research area of state space search for making a next move. 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 [5][6].

Section II briefly explains the history of Go-moku. Section III explains the rules of the game and various structures of the game and threat they provide to the opponent. Section IV provides brief introduction to genetic algorithm and fitness function evaluation. Section V onwards show how genetic algorithm is applied to Go-moku and fitness function evaluation with conclusion.

2. HISTORY OF GO-MOKU

Go-moku is an ancient Japanese strategic two-player board game. Go-moku (Five-in-line) is a specific form of a general game connect-X, where X=5.

Go-moku is a two-person, zero-sum, deterministic finite board game with perfect information. Two-person zero-sum games are characterized by the fact only one player wins, or it results in a draw.

In deterministic games, there is no blind move. It is not dependent on luck such as the roll of a dice. In addition to that Go-moku is a finite game, because there are a finite number of moves. As the board is visible to both players, it is also known as perfect information [7].

3. RULES OF THE GAME

The game is played on various sizes of boards such as 15*15, 17*17, 19*19 as is the case with game of GO. It is having very simple rules but is a highly complex game. The players alternate their moves. The players have unlimited number of pieces. One player plays with black colour pieces, and the other player with white colour pieces. Each move consists of putting a piece in the crossing points of horizontal and vertical line. The move can be made in any free position on the board. The player with black piece starts the game. The game is over when one of the player has got five pieces in one line either horizontally, vertically, or diagonally-major or minor.

The black normally plays the first move in the center of the board. Japanese professional Go-Moku players have stated for many decades that the player to move first i.e black normally wins, because the game provides advantage to first player [19]. To reduce that advantage, in a variant of the game, the next move of black is restricted in 5 * 5 square area of first move.

In Go-moku (Five-in-line), there are some structures which indicate a threat to the opponent, and the opponent needs to take some preventive or forced move to avoid loss. Following are some important structures [3].

3.1 Open four

It is some area on the board, where one of the players has already placed 4 pieces in a row, and wins if he places the next piece in line with existing 4 pieces. In this situation, the player can put a piece in any of the two positions, while the opponent can block only one position.

3.2 Four

This structure is very similar to open four except that; four pieces are open only on one side, while the other side is blocked by the opponent piece. The opponent has to make the next move in that position to avoid a loss and continue the game.

3.3 Three

In this structure, 3 pieces are in a row. If the three is open, then the player may convert it into open four. So, the player must take appropriate action.

3.4 Split three

In this type of structure, three pieces are in a row, but there is an empty place in between. Here, the player has a chance to make it into four. The opponent must prevent it by putting a piece in the middle or on any one side.

3.5 Game Complexity

State-space complexity of any board game represents the number of possible board states in the game. For example, in the game of Go-moku, there are 361 board locations where each location can take one of three values, giving approximately 3^{361} total state space.

4. GENETIC ALGORITHMS

There are many evolutionary algorithms. Genetic algorithm is the subset of evolutionary algorithms. It provides an algorithm and natural framework for exploiting all board game scenarios through natural evolution processes like selection, cross-over and mutation. So, it is a natural choice for game playing and learning problems. It is iterative in nature. In stead of using exhaustive search or conventional optimization techniques, it uses randomized searching. In practice, Genetic algorithms have been applied to a broad range of learning and optimization problems through a set of Genetic Parameters shown in the outline of genetic algorithm.

The program acquires a novel set of evaluation function parameters as generations of the genetic algorithms are executed through a series of experiments.

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population
3. **[New population]** Create a new population by repeating following steps until the new population is complete
4. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected) The idea is to choose the better parents.
5. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
6. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
7. **[Accepting]** Place new offspring in a new population
8. **[Replace]** Use new generated population for a further run of algorithm
9. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
10. **[Loop]** Go to step **2**

Outline of Genetic Algorithm

To begin the process of evolution, it starts with a random set of candidate solutions also called as chromosomes. These set of candidate solutions is known as population. Using a cross over process and mutation operators, it evolves the population towards an optimal set of solutions. The genetic algorithm does not give guarantee of optimal solution, so the main challenge is to design a “genetic” process that maximizes the likelihood of generating such an optimized solution [12].

In each generation, the fitness value of each candidate solution is evaluated, based on the fitness values, fittest candidate solutions act as parents of the next generation of candidate solutions. After being selected for reproduction process, parents are recombined or mated through a *crossover* operator and then mutated using a *mutation* operator to generate offspring. The fittest parents and their new offspring form a new population, from which the process is repeated to create new populations in the coming generations.

The design of evaluation function in genetic algorithm is domain specific. Selection, recombination, and mutation are generic operations in any genetic algorithm. The operations of evaluation, selection, recombination and mutation are usually performed repetitively for each of the iteration. So in a genetic algorithm, a major challenge is the design of the fitness function and the structure of chromosomes which reflects the problem domain. The value returned by the fitness function is called as fitness value. Other important parameters in Genetic algorithms are the size of the population, the portion of the population taking part in recombination, and the mutation rate. The mutation rate defines the probability with which a bit is changed in a chromosome that is produced by a crossover [13].

4.1 Fitness Function

Designing an efficient fitness function is the real challenge in genetic algorithm. The fitness function defines the fitness of each chromosome where the values of genetic parameters are adapted as the genetic evolution progresses. At every generation, fitness value of each chromosome is calculated using fitness function. In order to avoid the problem of local minima or local maxima, If fitness of two chromosomes is equal, then the mutation rate is increased. Once there is an improvement in the overall fitness, the original mutation rate is restored to continue evolution as normal. If the evolution stabilizes, but the fitness does not seem to be improving for several generations and the search does not find any error, new set of initial population is generated using the initial default parameter values and a new randomly generated seed [14].

This paper uses genetic algorithms in game of Go-moku by constructing a static evaluation function based on the features and strategies of the game.

5. APPLYING GENETIC ALGORITHM TO GO-MOKU

Go-moku was chosen because the rules of the game are easy to implement, the game provides reasonable complexity and the game is guaranteed to finish. So, Go-moku ia a perfect game for optimizing using genetic algorithm.

The initial population is randomly generated and a random evaluation function serves the purpose of finding fitness value.

The board is represented as two dimensional array of 19 * 19 size. The computer plays using GA. Each element can take one of the following 4 values: -1= Free position in neighboring zone, 0 =Free position, 1= Computer player (using GA) piece, 2= Human player piece [3].

The neighboring zone of considered board position is the set of all occupied positions i.e. positions with value 1 or 2. We use a representation for a move which is represented by 3 integers, namely x, y and fitness value, where x and y are horizontal and vertical positions of the board. The fitness value is derived using fitness function for a move to position (x, y) on the board. The chromosome represents a sequence of alternate plays by computer algorithm and player.

Figure 1 shows the structure of chromosome. The chromosomes cannot contain equal genes and the genes must be placed in the free position of the board.

x1	y1	f1	x2	y2	f2	x3	y3	z3

Fig 1: structure of chromosome

The fitness function uses a table of weights to calculate the fitness value for a considered board position. The fitness value of a gene is calculated as the sum of weights of all sequences of pieces surrounding the gene under consideration. The static weights used are as mentioned in the Table1.

Table 1.The static weights

Number of pieces	1	2	Three structure	Four structure
Computer Play (GA)	4	16	96	4800
Human Player	2	10	80	2900

For the figure 2, the fitness value according the Table- 1 will be calculated as per the neighboring pieces of the considered cross (X) mark and it will be 16 + 10 =26.

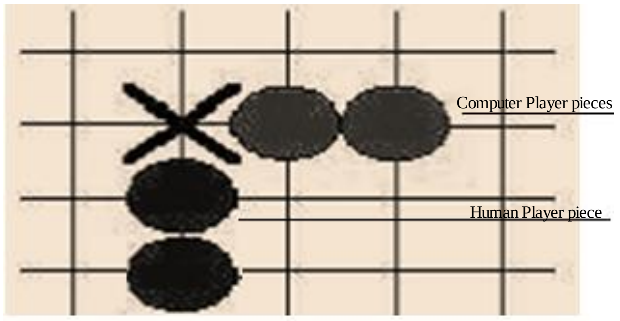

Fig2: fitness value

So the most important question here is the "weight" of evaluation function. In this case, finding out correct weights is the key. Genetic algorithms learn the good weights as the program proceeds. The way to do this is to start with a random set of weights in the program, and use them to test the program. If the program does well, we keep the weights, and use those (making small changes) in the next version of the program. If they do badly, they are discarded, and start again with a new set [3]. To produce feasible chromosomes, crossover operation can use heuristics also. After crossover operation, the genes are sorted according to their fitness values. So, the first gene in the chromosome has the highest fitness value and the second gene in the chromosome represents the move by the opponent. The next move by computer is calculated depending upon the prediction value.

The algorithm calculates the sum total of first genes which occur as per the prediction value selected. If the prediction value is 3, then sum total of first 3 genes take place. This value is used to decide the next actual move suggested by the first gene of selected chromosome. The fitness value guides the search for the next move, but is not the only criteria. In some situations, where defense is important to avoid loss of game, the move should be selected which gives more preference to defense and not to attack.

Depending on the problem, we may be interested in minimizing or maximizing the value of fitness value. For the Go-Moku problem, we want to maximize the value of fitness.

6. GAME PROGRAM IMPLEMENTA-TION

The chromosome is represented as a structure with three variables x, y and fitness value for that position. The weights for the various structures like open four, four, three and split three etc. are calculated as per the structures and position of the pieces on the board. The fitness value of the considered board position is calculated as the sum total of the weight values of the surrounding genes in the neighborhood. According to the fitness values found for all considered board positions, the move having the maximum fitness values is selected as the next move by computer. Whenever required the program gives more preference to defense than attack in order to extend the game and delay or avoid loss [3].

7. EXPERIMENTS AND RESULTS

In the experiment, following parameters were taken for the genetic algorithm. The default values set are shown in Table- 2. Most important feature is number of genes in the chromosome and also the size of population. The Prediction parameter, which shows how many steps forward look ahead affects the computing time of genetic algorithm.

Table 2.Parameters for GA

Name of Parameter	Default value
Number of genes per chromosome	25
Population size	25
Prediction	3
Rate of crossover	0.5
Number of iterations (generations)	15

The used weights influence the quality of the moves of the program. To test the performance of the algorithm, one set of 15 consecutive human-computers moves were taken.

As shown in figure 3, With genes per chromosome=20 and population size =20, on average the GA had 3.27 and 3.33 times blocked the sequences of 3 and 4 pieces of opponent respectively.

The results showed that on average the GA attacked more than defended in the game. If we interchange first and second row of the table, the GA would use the strategy of defense instead of strategy of attack. Using these parameters, the GA won 68% of the time as against 60% of the time as compared to [3].

The execution time for each move is below hundredth of a second for the default parameters set using a Pentium III processor with, 2 GB RAM, under Windows XP. The paper shows how we could implement a board-game without using the search tree or game-tree.

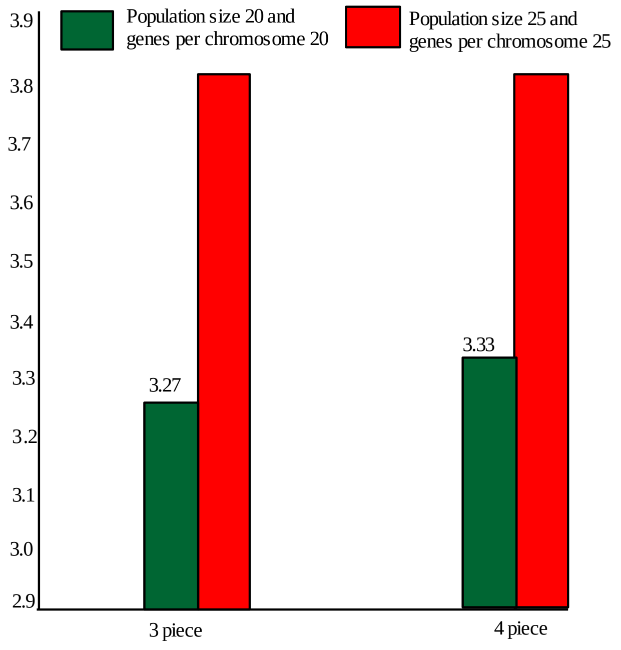

Fig. 3: Average the GA

8. CONCLUSION

The simplicity of fitness function is heavily based on the feature characteristics of the game. The analysis and construction of features is the main driving force to solve the game in terms of creating fitness function. This function when passes through the genetic cycle of selection-crossover-mutation with weight tuning through iterative process of generations it exposes a possibility of improvement and some rearrangement of weights to produce brilliant moves for attack and defense strategies.

This implementation, which takes moderate number of Genetic Algorithm constituents like Number of Genes in Chromosome, Population size, Number of Generations, not only improves the working cycle of better game moves, but also show very promising side of Genetic move optimization.

As the number of genes in a chromosome and population size increases, the GA plays in a better way and blocks the opponents 3 and 4 piece sequences efficiently as compared to as shown in [3].

9. 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]. Shah Sanjay M , Singh Dharm, Thaker Chirag S. Multimedia Based Fitness Function Optimization Through Evolutionary Game Learning

[4]. J¨org Denzinger, Kevin Loose, Darryl Gates, and John Buchanan. Dealing with parameterized actions in behavior

testing of commercial computer games. In Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG), pages 37–43, 2005.

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

[6]. 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.

[7]. 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.

[8]. Thomas P. Runarsson and Simon M. Lucas. Co-evolution versus self-play temporal difference learning for acquiring position evaluation in small-board go. IEEE Transactions on Evolutionary Computation, 9:628 – 640, 2005.

[9]. Yannakakis, G., Levine, J., and Hallam, J. (2004). An evolutionary approach for interactive computer games. In Evolutionary Computation, 2004. CEC2004. Congress on Evolutionary Computation, volume 1, pages 986–993, Piscataway, NJ. IEEE.

[10]. A. Hauptman and M. Sipper. Evolution of an efficient search algorithm for the Mate-in-N problem in chess. In Proceedings of the 2007 European Conference on Genetic Programming, pages 78–89. Springer, Valencia, Spain, 2007.

[11]. P. Aksenov. Genetic algorithms for optimising chess position scoring. Master's Thesis, University of Joensuu, Finland, 2004. Y. Bjornsson and T.A. Marsland. Multi-cut alpha-beta-pruning in game-tree search. Theoretical Computer Science, 252(1-2):177–196, 2001.

[12]. O. David-Tabibi, A. Felner, and N.S. Netanyahu. Blockage detection in pawn endings. Computers and Games CG 2004, eds. H.J. van den Herik, Y. Bjornsson, and N.S. Netanyahu, pages 187–201. Springer-Verlag, 2006.

[13]. A. Hauptman and M. Sipper. Using genetic programming to evolve chess endgame players. In Proceedings of the 2005 European Conference onGenetic Programming, pages 120–131. Springer, Lausanne, Switzerland, 2005.

[14]. G. Kendall and G. Whitwell. An evolutionary approach for the tuning of a chess evaluation function using population dynamics. In Proceedings of the 2001 Congress on Evolutionary Computation, pages 995–1002. IEEE Press, World Trade Center, Seoul, Korea, 2001.

[15]. Holland, J. H. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. Ann Arbor, MI: University of Michigan Press.

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

[17]. Buckles Bill P. and Petry, Frederick E. Genetic Algorithms. Los Alamitos, CA: The IEEE Computer Society Press. 1992.

[18]. Haupt, Randy L, and Haupt, Sue Ellen. (1998). Practical Genetic Algorithms. New York: John wiley & Sons

[19]. L.V. Allis,H.J. van den Herik ,M.P.H. Huntjens. Go-Moku and Threat Space Search.

[20]. Sanjay Shah, Dharm Singh, Chirag S. Thaker, Multimedia Based Fitness Function Optimization Through Evolutionary Game Learning., 2011 ETNCC, pp 164-168, IEEE Catalog Number CFP1196N-CDR , ISBN 978-1-4577-0238-9 and IEEE Catalog Number CFP1196N-ART , ISBN 978-1-4577-0240-2.