International Journal of Modern Trends in Engineering and Research
www.ijmter.com
e-ISSN: 2349-9745

Optimizing Search Space of Othello Using Hybrid Approach

Chetan Chudasama[1], Pramod Tripathi[2], keyur Prajapati[3]
[1]*Computer Engineering, MBICT, New V. V. Nagar, chetanfriends30@gmail.com*
[2]*Computer Engineering, MBICT, New V.V.Nagar, ptripathi@mbict.ac.in*
[3]*Computer Engineering, MBICT, New V.V.Nagar,kprajapati@mbict.ac.in*

Abstract—one of the areas of Artificial Intelligence is Game Playing. Game playing programs are often described as being a combination of search and knowledge. The board games are very popular. Board games provide dynamic environments that make them ideal area of computational intelligence theories. Othello is 8 × 8 board game and it has very huge state space as $3^{64} \approx 10^{28}$ total states. It is implemented by game search tree like Mini-max algorithm, alpha-beta pruning. But it required more storage memory and more time to compute best move among all valid moves. Evolutionary algorithms such as Genetic algorithm are applied to the game playing because of the very large state space of the problem. Game search tree algorithm like alpha- beta pruning is used to build efficient computer player program. This paper mainly highlights on hybrid approach which is combination of Genetic algorithm and alpha-beta pruning. Genetic algorithm is applied to optimize search space of Othello game and building Genetic Weight Vector. This weight vector is applied to game which played by alpha- beta pruning game search tree algorithm. And we optimize search space of Othello and get best move in less amount of time.

Keywords-Game Playing, Board Game, Genetic algorithm, Othello, Alpha- Beta Pruning

I. INTRODUCTION

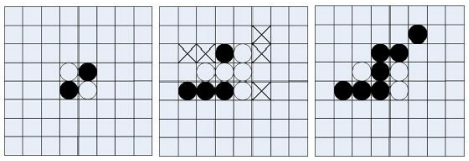
Games have long dominated popular area of artificial intelligence (AI) research, and for all good reasons they are very challenging on one part and yet easy to formalize and develop new AI solution exploration methods. Through games efficiency of AI working can be measured in terms of capability to acquire intelligence without putting human lives or property at risk. Most research based work so far has focused on games that can be described and solved-won in a compact form using symbolic representations, such as board and card games. The so-called "good old-fashioned artificial intelligence" techniques work well with them, and to a large extent, such techniques were developed, tested and improvised for such games [1][2].

Almost all traditional board games involves great amount of AI research as they all has very high space complexity to traverse before making good moves. These games poses main challenges in form of guiding the evolution using human knowledge and in achieving game playing behavior that is not only successful, but visibly intelligent to a human observer [3][4].

So, Games like chess and Othello have been subjects of research for many years. These games are interesting because of their complexity and their many possible game states while they are also fully observable and deterministic. And because of a well defined set of rules they are easy to implement.

II. INTRODUCTION OF OTHELLO

Othello is one of the prominent and traditionally very popular in Japan played since centuries. It was brought to European countries and America in the mid 1970's. A line for the game talks about it very perfectly as it says "A Minute to Learn A Lifetime to Master". So beginners and experts enjoy it initially for its simple playing rules, but complex strategies must be mastered to play the game well [4]. It is also known as Reversi.



(A) (B) (C)

Fig. 1 (A) Initial State of Board (B) After Five legal moves- (the legal moves for black are marked with X's). (C) After black has moved to the rightmost X.

2.1 Rule of Game

Othello is a two-player game played on an 8 X 8 board. It is a two-player, deterministic, zero sum, alternative move and perfect information game. While studying the game board characteristics it is found that the game is having symmetry phenomenon which helps in dividing the 8X8 board (which are sixty four disc blocks) into ten disc sets. The goodness is that all board-discs (or pieces) are identical with one white side and one black side. The initial board setup is very well presented in Figure 1(A). Each player takes turns placing discs on the board with his color face up.

A player is only allowed to move in an open space that causes an opponent's disc or discs to be ranked by the new disc and another one of the player's own discs. Discs may be captured vertically, horizontally, or diagonally. Figure 1(B) shows the legal moves for black for the given board pattern. Once a move is made, the captured discs are flipped. Figure 1(C) shows the board layout resulting from a move by black in the sixth row of the sixth column. The game is continued until there are no legal moves available for either player. If a player has no legal move, he must pass. The winner is the player with the most on board discs in the final board configuration [5] [6].

2.2 Game Complexity

State-space complexity of any board game represents the number of possible board states in the game. For example, in Othello there are 64 board locations where each location can take one of three values, giving approximately $3^{64} \approx 10^{28}$ total states. Game tree complexity represents the total number of nodes in a fully-expanded game tree. Othello has a game tree complexity of 10^{58}. Based on that figure, the average branching factor is about 9.26. To check this, we made two random players play against each other, recording the number of available moves for each stage of the game. The branching factor rises slowly from 4 and peaks at 12 on the 30^{th} move [7].

III. GENETIC ALGORITHM

Genetic algorithm being an effective and efficient subset of evolutionary algorithms is a natural protestor for learning in games. A genetic algorithm provides an algorithmic and logical framework for exploiting all possible board state scenarios through natural-evolution processes like selection, crossover and mutation. It evolves intermediate candidate solutions to problem domains that have large solution search spaces and are not open to exhaustive search or conventional optimization techniques. Since ages Genetic algorithms have been applied to a broad range of learning and optimization problems [8][9][16].

When genetic evolution process is run for many numbers of generations through a series of experiments, the program acquires a novel set of evaluation function parameters are gathered for subsequent population. Normally, a genetic evolution process starts with a random set of population candidate solutions, called chromosomes. Through a cross over process and mutation operators, it evolves the population towards an optimal solution. GAs does not guarantee an optimal solution hence the challenge is to design a "genetic" process that maximizes the likelihood of generating such an optimized solution [10][14].

The first step is typically to evaluate the fitness of each candidate solution in the current population based on the fitness values and to select the fittest candidate solutions to act as parents of the next generation of candidate solutions. After being selected for reproduction, parents are recombined 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. This procedure is shown in figure 2.

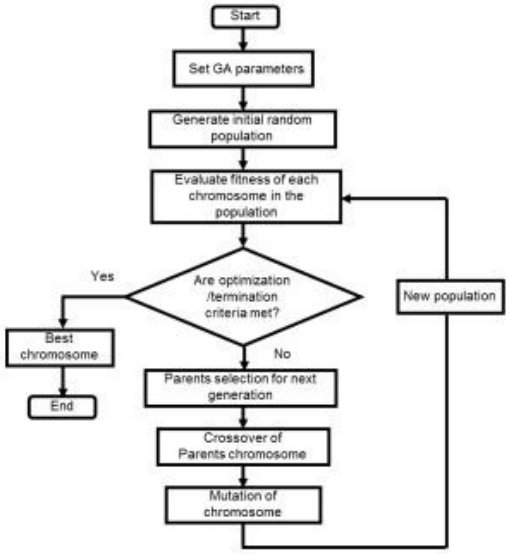


Fig 2. Procedure of Genetic Algorithm

The operations of evaluation, selection, recombination and mutation are usually performed many times in a genetic algorithm. Selection, recombination, and mutation are generic operations in any genetic algorithm. Evaluation is problem specific and relates directly to the structure of the solutions So in a genetic algorithm, a major issue is the choice of the structure of solutions and of the method of evaluation (fitness function).Other important parameters include 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 flipped in a chromosome that is produced by a crossover [11].

3.1 Fitness Function

An important parameter of a genetic algorithm is the fitness function that defines the fitness of each chromosome where the values of genetic parameters are adapted as the genetic evolution progresses. At every generation the best and worst chromosome fitness are kept track of. In some cases if both fitness values become equal, 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. Here if evolution stabilizes by the fitness does not seem to be improving for

several generations and the search does not find any error, the genetic algorithm with the initial default parameter values and a new randomly generated seed to generate a new random initial population [12][15].

This paper uses genetic algorithms in game-playing learning programs by constructing a static evaluation function based on the features and strategies of Othello. To attain better results we modify genetic algorithm as per the complexity of evaluation function parameters.

IV. ALPHA- BETA PRUNING

Alpha-beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the mini-max algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard mini-max tree, it returns the same move as mini-max would, but prunes away branches that cannot possibly influence the final decision [14].

The benefit of alpha-beta pruning lies in the fact that branches of the search tree can be eliminated. This way, the search time can be limited to the 'more promising' sub tree, and a deeper search can be performed in the same time. Like its predecessor, it belongs to the branch and bound class of algorithms. The optimization reduces the effective depth to slightly more than half that of simple mini-max if the nodes are evaluated in an optimal or near optimal order (best choice for side on move ordered first at each node).

Normally during alpha-beta, the sub-trees are temporarily dominated by either a first player advantage (when many first player moves are good, and at each search depth the first move checked by the first player is adequate, but all second player responses are required to try to find a refutation), or vice versa. This advantage can switch sides many times during the search if the move ordering is incorrect, each time leading to inefficiency. As the number of positions searched decreases exponentially each move nearer the current position, it is worth spending considerable effort on sorting early moves. An improved sort at any depth will exponentially reduce the total number of positions searched, but sorting all positions at depths near the root node is relatively cheap as there are so few of them. In practice, the move ordering is often determined by the results of earlier, smaller searches, such as through iterative deepening.

The algorithm maintains two values, alpha and beta, which represents the maximum score that the maximizing player is assured of and the minimum score that the minimizing player, is assured of respectively. Initially alpha is negative infinity and beta is positive infinity. As the recursion progresses the "window" becomes smaller. When beta becomes less than Alpha, it means that the current position cannot be the result of best play by both players and hence need not be explored further. pseudo code of Alpha-Beta pruning shown below.

Alpha- Beta Pruning Pseudo code [14]

```
function alphabeta(node, depth, α, β, Player) if
  depth = 0 or node is a terminal node
    return the heuristic value of
  node if Player = MaxPlayer
    for each child of node
      α := max(α, alphabeta(child, depth-1, α, β, not(Player) ))
      if β ≤ α
        break (* Beta cut-off *) return α
```



```
else
  for each child of node
    β := min(β, alphabeta(child, depth-1, α, β, not(Player)
    )) if β ≤ α
      break (* Alpha cut-off *) return β

(* Initial call *)
alphabeta(origin, depth, -infinity, +infinity, MaxPlayer)
```

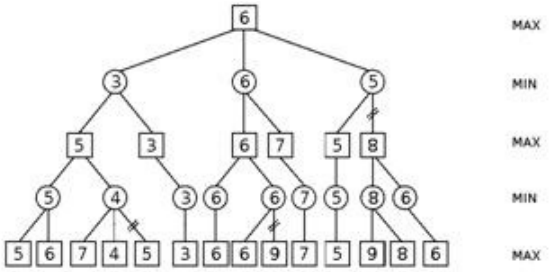


Fig 3 Illustration of Alpha-beta pruning

An illustration of alpha-beta pruning is shown in figure 3. The grayed-out sub trees need not be explored (when moves are evaluated from left to right), since we know the group of sub trees as a whole yields the value of an equivalent sub tree or worse, and as such cannot influence the final result. The max and min levels represent the turn of the player and the adversary, respectively.

V. PROPOSED APPROACH

A Here Hybrid Method Approach is applied to optimize search space of Othello game. The hybrid approach contains two algorithms one is Genetic Algorithm and other is Alpha-Beta pruning game tree searching algorithm. Genetic Algorithm is applied at pre- processed step of Othello game. Then Game is played by alpha-beta pruning algorithm.

Steps of proposed approach

1. Set Genetic Algorithm parameters like Cross Over rate, mutation rate, population size, No. of generation.
2. Encode game parameter by suitable encoding methods; select appropriate parent selection method and build fitness function.
3. Build Genetic Weight Vector by applying Genetic Algorithm to Othello game according to board positional weights.
4. Use Genetic Weight Vector and Game positional weights that shown in figure 4 in Alpha-beta pruning algorithm to find the optimal or best move among all legal moves for computer player.

0	1	3	6	6	3	1	0
1	2	4	7	7	4	2	1
3	4	5	8	8	5	4	3
6	7	8	9	9	8	7	6
6	7	8	9	9	8	7	6
3	4	5	8	8	5	4	3
1	2	4	7	7	4	2	1
0	1	3	6	6	3	1	0

Fig. 4 Board position parameters

VI. IMPLEMENTATION

Value encoding strategy was used to build chromosomes, these shows the board position and its parameters according to players. At every generation fitness value of each chromosome is calculated using fitness function. Here f was fitness function that is taken for implementation.

Fitness function f = W1 × Y1 + W2 × Y2 +…. + W10 × Y10

Here W is Genetic Weight Vector

Y is Game position weight vector according player discs that are placed at different position on the board.

Population size was set to 20. Generations were generated up to 32. The crossover rate and mutation rate is taken 45% and 0.2% respectively. Tournament Selection method was selected as parent selection method for selecting parent for next generation.

VII. RESULT ANALYSIS

We applied to genetic algorithm at prior stage of game and build Genetic Weight Vector. This gives best move among all possible legal moves comparatively less amount of time than applying only alpha-beta pruning algorithm and optimize the search space of Othello Game. This observation is shown in figure 5. This shows that at initial stage of game both the approach is take same amount time but as game is played proposed hybrid approach is take less amount time.

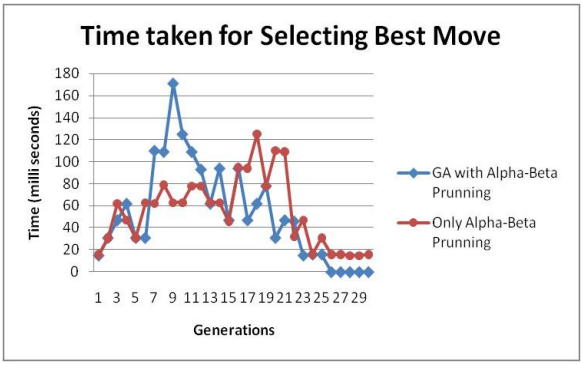


Fig. 5 Time comparison ordinary method and proposed approach

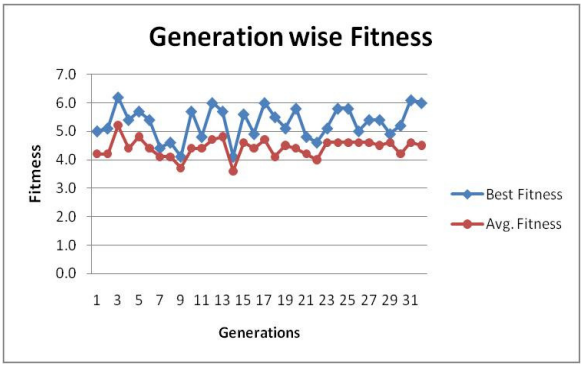


Fig. 6 Generation wise fitness

Figure 6 shows the max and average fitness of chromosomes in each generation. This shows that at starting stage fitness value is smaller and at later stage average fitness is constant and max fitness is increase.

VIII. CONCLUSION

Implementation of Genetic Algorithm in Othello Game in early stage to build the evolution weight vector and this evolution vector helps in performance (percentage of games won), speed, predictability of opponent, and usage situation. Using of Genetic Algorithm and alpha beta search technique computer player selects Best move from number possible valid move. Increase points to win the Game and defeat the opponent player. This proposed technique is better than existing simple alpha-beta pruning.

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] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Russell, Stuart J.; Norvig, Peter (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Upper Saddle River, New Jersey: Pearson Education, Inc.. p. 167. ISBN 0-13-604259-7

[14] Dharm Singh, Chirag S Thaker, Sanjay M Shah, Quality of State Improvisation Through Evaluation Function Optimization in Genetic Application Learning, 978-1-4577-0240-2/11/$26.00 ©2011 IEEE, pages 98- 97.

[15] Dharm Singh, Chirag S Thaker, Sanjay M Shah, Fitness Value Optimization for Disc Set in Board Game , © 2011 by IJCA Journal, pages- 1-6

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

