Performance Improvement in Game Playing using Evolutionary Computation by Large Search Space Exploration

Chirag S Thaker, Sanjay M Shah
Department of Computer Science & Engineering
Sursesh Gyan Vihar University
Jaipur (India)
chiragthaker@yahoo.com, sanjay_shah_r@yahoo.com

Dharm Singh
College of Technology and Engineering
Maharana Pratap University of Agriculture and Technology
Udaipur (India)
dharm@mpuat.ac.in

Abstract— **Soft Computing branch of intelligence research is primarily focused on the path of achieving high performance by mimicking the human approach. The central idea is to capture and encode human knowledge in artificial learning form. Applying AI technology to develop efficient game-playing programs is through realization of search-intensive approach in very large and complex search intensive areas. Many researchers have developed very powerful search techniques over the past two decades and successfully applied these search algorithms to problems domains of optimization, machine learning and soft computing paradigms. This paper extends this approach, by developing a program which is almost completely reliant on search optimization through evolutionary computation. Very efficient evolutionary algorithms and advancement of "intelligent" search along with improved hardware resources like faster processors, larger memories, and larger disks makes it possible to push the limits to solve problem of type and size of Checkers which has search space as high as 5×10^{20} representing a daunting challenge. The collected checkers result pushes the boundary of evolutionary algorithms based problem domains.**

Keywords— Soft Computing, Evolutionary Algorithm, Search, Checkers, Generation.

I. INTRODUCTION

Since centuries games of brain-skill is an important fascination domain for people because of its unique nature of intellectual challenge and satisfaction which is derived from playing well. Many games have too many possibilities for a human to understand before making any valid–sharp move at any given stage. Since last fifty years of research in the area of computer games, many powerful learning methods have evolved to use knowledge and search to make game playing decisions on the board. The human or computer "player" with the best game playing program or algorithm wins in the long run. Without perfect board rules, knowledge and decision making skill mistakes are made, and results in game loss for experts also [1]. Any board-game in general and Game of Checkers in particular will definitely push the boundary of AI research through its collected results. Since ages of AI research evolution, the first and foremost easy path to achieve high performance is nothing but to imitate the human approach. However this was weighed down with difficulty of acquiring, interpreting and encoding human knowledge. It is learnt that human-like game playing or "intelligent" move making strategies are not necessarily the best computational strategies in terms of implementing and collecting results [2].

There is a lot to "learn" and "understand" through algorithms and programs while implementing any problem which requires "knowledge" and "decision making process". In fact, one of the leading contributions of applying AI expertise to develop game-playing programs was the realization that a search-intensive ("brute-force") approach. This has potential of producing high-quality performance using minimal domain-specific knowledge. Due to consistent efforts made by AI research groups all over the world, very powerful and result providing search techniques have been developed and successfully deployed to variety of problems areas like optimization, planning, and bioinformatics [3]. The application domain of game learning and move making programs are primarily an optimization problem. Here each program carries out a search, with current board position as root node(s). The degree of sophistication lies in efficiency of search algorithms which in turn evaluates current power of its evaluation function in very large search space. The game playing programs have two major dimensions of consideration:

- Decision complexity, the difficulty of making correct move decisions,
- Space complexity, the size of the search space.

Game of Checkers is considered to have high decision complexity as it requires extensive skill to make strong move choices and moderate space complexity (5×10^{20}) [4].

II. EVOLUTIONARY ALGORITHM

This evolutionary algorithmic approach to machine learning offers the potential for exploring any possible solution corridor presented by a computational complex problem. It does not aim to complete an exhaustive search, but it has power to quickly identify and converge on useful solutions. It provides an effective means for going beyond the structured conventional

engineering approach which is very common to many forms of human design. The solution search process is inherently parallel and can be accelerated significantly by utilizing an evolutionary search algorithmic to find fitter solutions through strong moves. At a stage, all that is required is to be able to compare two solutions and indicate which is better. The process of Darwinian evolution can then optimize the solution for the given problem. In this mode, evolutionary machine learning can meet and eventually go beyond the capabilities of human expertise [5].

In evolutionary algorithm, the proposed solution does not aim to find global optimum. The program's main objective lies in tuning an evaluation function to adjust its parameters so that the overall game performance of the program is enhanced. In fact, for a large set of such high complexity driven problems, a unique global optimum does not exist. At first glance, automatic tuning of the evaluation function appears like an optimization task, which is very well suited for Evolutionary algorithms like GA. The many parameters associated with the evaluation function which are nothing but the mirroring of the features associated with the game, can be encoded as a bit-string.

This initial set of bit-string can be randomly initialized in form of "chromosomes", each one representing one evaluation function. Then after, the population is evolved through many generations until a highly tuned "fit" evaluation function gets emerged. This evolutionary process has one major blockage that hinders the application of GA, namely the fitness function. For a given a set of evaluation function parameters, encoded as a chromosome, the main objective is to calculate the fitness value. For many research years, firsthand solution was to let the individuals in each generation was allowed to play against each other a series of program or algorithmic operations. Then subsequently, take the fitness score of each individual in each of the generations. The main shortcoming of this approach is the unacceptably large amount of time needed to evolve each generation. As a result, it imposes severe limitations on the length of the iterations played after each generation, and also on the size of the population involved.

Game of Checkers relies on features that were very well chosen using human expertise and weighted by evaluation function tuning. It is completely dependent on game moves' database for all possible moves explored through extensive game board possibilities. Once the best move exploration is reached, Computer Checkers will never make a mistake because the final outcomes that can be achieved from these states have already been evaluated. Game of Checkers aim to improve its play through game rules and perfect information of board positions which gives high-speed evolutionary computation to look ahead as many ply (search depth) as possible [6].

The evolved evolutionary program exhibits a flexibility that can be achieved with Checkers playing program in evolutionary approaches which is heavily dependent on all of their "intelligence" being pre-programmed. The evolutionary algorithm is also capable of adapting its game play to meet more demanding challenges from better-quality opponents. It can conceive new and untraditional procedures.

Evolutionary algorithms explore a large number of points simultaneously in a search space. This phenomenon avoids the chances of poor local optima quickly, thus resulting in a quicker and fruitful search. By coding a given Checkers problem into an Evolutionary computation framework, these smart algorithms are able to "evolve" solutions to real world problems [7]. The game of checkers has roughly 500 billion-billion possible positions (5×10^{20}).The task is very daunting to solve the game, determining the finishing result in a game with no error made by either of the player. Since last three decades, almost incessantly, dozens of computers have been working on solving Game of Checkers, applying state-of-the-art soft computing based techniques to improve the learning process [8].

A. Game Complexity

The game of checkers has roughly 500 billion- billion possible positions (5×10^{20}). The task is very daunting to solve the game, determining the finishing result in a game with no error made by either of the player. Since last three decades, almost incessantly, dozens of computers have been working on solving Game of Checkers, applying state-of-the-art soft computing based techniques to improve the learning process [9]. Game of Checkers represents the most computationally challenging game to be solved to date. Evolutionary Learning challenges in Game of Checkers are:

(1) The space to be searched is huge. It is estimated that there are up to $5X10^{20}$ possible positions that can be searched. So any search algorithm based method which is based on exhaustive search for the problem space is infeasible.

(2) The search space volume is not smooth and straight forward. An evaluation function's parameters which is feature construction based are highly inter-dependent. In some cases increasing the values of optimization parameters will result in a worse performance, but many a times the controlled set of evolutionary parameter is also increases performance, then an improved overall performance would be obtained.

(3) The problem is not well understood by researchers. Even though all top performing programs parameters are hand tuned by their program designers, finding the best value for each parameter is mostly based on operational genetic alternatives [10][11].

III. INTRODUCTION TO CHECKERS

Checkers (also known as "draughts") is played on an eight-by-eight board with squares of alternating colours. There are numerous variants (more than 150 worldwide) of the game played around the world. There are two players who take sides denoted by "red" or "white" (or "black" and "white").

Each side has 12 pieces which are also called checkers that begin in the 12 alternating squares of the same colour that are closest to that player's side, with the rightmost square on the closest row to the player remaining open (Figure 1) [12].

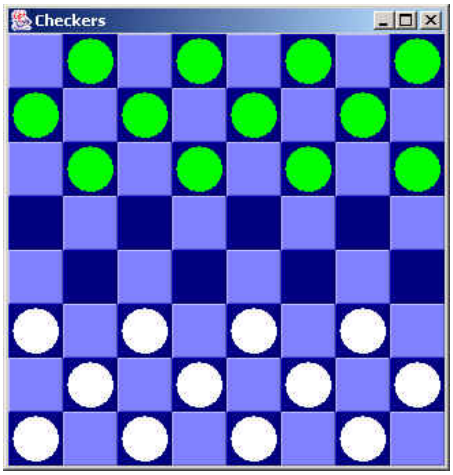

Figure 1. Checkers Initial Board

Checkers move one square at a time forward diagonally or, when possible, may jump over an opposing checker into an empty square. This jump moves are forced, although if more than one possible jump is available, the player may choose which jump to execute. When any checker advances to the last row of the board it advances to become a king and now may move forward or backward diagonally.

The game ends when any one side cannot make a legal move, which is most commonly brought about by removing that player's final piece. The player who cannot move loses and the other player wins. Alternatively, a draw may be declared upon mutual agreement of the players, or in tournament play at the discretion of a third party under certain circumstances. In some tournament play, a time restriction (60 minutes for the first 30 moves) is also introduced which if violated results in a loss. This paper is restricted to the 8*X*8 variant, but many of the ideas presented here also apply to the 10 *X* 10 game [13][14].

The game of Checkers has one very distinguishing and firm game feature that there is no "centre space" on the game board that each player from the starting position is equally likely to occupy. One very interesting and thought provoking facet of the game is the strategy involved in moving and capturing with promoted pieces. A piece may be promoted if its move *ends* on a square located in the opponent's back row. A piece would not be promoted if during its move it travels through the back row but continues moving and finally settles on a space in a row other than the back row then that piece is called "king" and counted as a promoted piece. The promoted pieces introduced a very interesting new strategic element.

- A promoted piece can move one space when not capturing.
- A promoted piece can make multiple captures in a single turn.
- A captured piece is considered removed from the board as soon as the piece capturing it passes over it. This allows promoted pieces to move back and forth along a diagonal to make multiple captures if possible.

They in fact increase forward and attacking activity, as both players attempted to gain an advantage by increasing their number of promoted pieces as quickly as possible. In general the strategic game players will always try to increase the frequency with which pieces were promoted, but this was not always the case as the players at times would simply avoid moving the pieces from their back two rows in order to prevent the opponent from promoting pieces. Now this defensive strategy creates an interesting method in which players would seem to attack in two tier wave pattern. In first round of wave they try to forward their pieces, and then after eventually as and when required a second wave of the pieces try to take their forward positions from back rows.

Here if one player is able to gain a number of promoted pieces early, it provides the player noticeable attacking advantage for that player and would create fair chance strength inequality which can prove to be very challenging for the second player to recover from. This inequality is not a regular phenomenon but the player who attains it by an "intelligent" game of play certainly enjoys it and sometimes proves to be a stepping platform to win the game of Checkers.

IV. CHECKERS IMPLEMENTATION

For the present research experimentation, each checkerboard was represented by a vector with length of 32, with each one side component corresponding to an available position on the board. Vector components were elements in the range from $\{-K, -1, 0, +1, +K\}$, where 0 corresponded to an empty square, 1 was the value of a regular checker, and K was the number assigned for a king. The sign of the value for component wad indicative for the piece belongingness, either to the player (positive) or the his/her opponent (negative).

A player's move was determined with evaluated parameters of the board game positions and their related vector weight values. The evaluation function was formulated as an evolutionary parameter values. The first vector parameter set was formed using experienced random set of values designed to indicate the spatial characteristics of the checkerboard. The remaining values are computed based on evolutionary computation parameters which are the dot product of the evolvable set of weights and their respective board square vector weight set. Each value has partial dependence on each preceding node [15].

When a initial board set up was formed and prepared for round of evolutionary algorithmic tests, the complexity enhancement due to board weights and holdings were interpreted as the worth of the board from the perspective of the player whose pieces were denoted by positive values. The move which is closer to the final output was to 1.0, the better the evaluation of the corresponding board. But if the move has potential to bring the output to –1.0, the board square situation is worsening which may result in combined loss over a long run of moves. All winning positions that were wins for the player (e.g., no remaining opposing pieces) were assigned a value of exactly 1.0, and similarly all losing positions were assigned a value of –1.0.

Each "parent" generated an off springs by varying all of the associated weights and related square values. All parents and offspring competed for survival by playing games of checkers and receiving points for the results of their play. Each player scored –2, 0, or +1 points for a loss, draw, or win, respectively. These values were chosen as being reasonable and no optimality is claimed. A draw was declared if a game lasted for 100 moves. In total, there were 150 games per generation, with each checker piece participating in an average of 7-10 games. After all games were complete, the 15 checkers squares that received the greatest total weight points were retained as parents for the next generation and the process was iterated [16].

Each game was played using a min-max alpha-beta search of the associated game tree that results from looking ahead over a selected number of moves. The ply depth of the search, *d*, was set at four (i.e., two moves for each side) to allow for reasonable execution times (30 generations). The depth ply was extended in units of two any time a forced jump move was encountered because in these situations no real decision is available.

The best move to make was chosen by iteratively minimizing or maximizing over the leaves of the game tree at each ply according to whether it was the opponent's or the player's move.

V. COLLECTED RESULTS

Over 200 games against human opponents were played with evolutionary algorithmic to determine best moves. No opponent was told that they were playing against a computer program, Games were played mainly using a ply of $d = 4$ to ensure reasonably fast play (i.e., under one minute per move, typically).

With a population size of 100 and by an maximum limitation of 1 minute per move for each player side, assuming each individual plays at least 10 games, it would take 2000 minutes for each generation to evolve. With these time constraints, reaching the 25th generation would take as long as few hours to complete the evolutionary algorithm based game of Checkers to evolve as shown in Fig 2 [17].

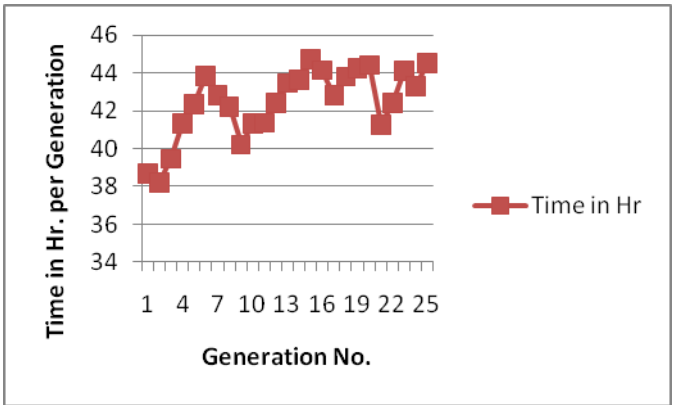

Figure 2. Time Analysis

Before that, we briefly review previous work in applying evolutionary methods in computer Game of Checkers. Evolutionary Genetic Computation was successfully employed by Checkers parameters used in their evaluation function. Their evolved program managed to compete with strong programs only if their search depth (look-ahead) was severely limited to ply =4.

Apart from the special fitness function described above, standard implementation of GA was also employed with proportional fitness based selection and single point crossover. The evolutionary genetic parameters are as follows.

- population size = 1000
- crossover rate = 0.75
- mutation rate = 0.002
- number of generations = 300

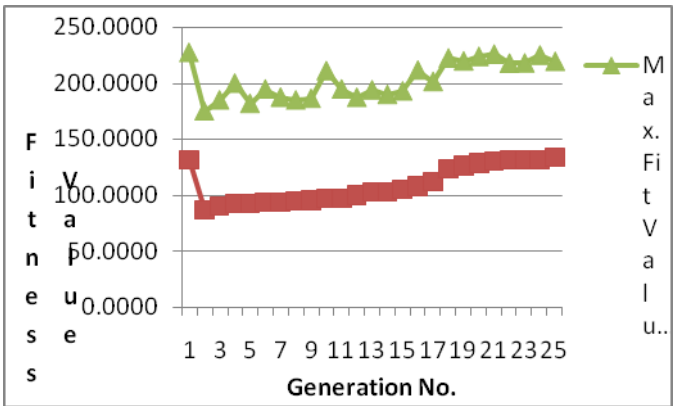

Figure 3. Fitness Value Analysis for sample 25 Generations

Apart from the special fitness function described above, we used a standard implementation of GA with proportional selection and single point crossover. The following are the parameters we used population size = 100, crossover rate = 0.75, mutation rate = 0.002 and number of generations = 25.The results are collected for Max. And Minimum Fitness value collected over a span of 25 generations and shown in Fig.3.

Here GAs as Evolutionary Algorithms were used for evolving evaluation function parameters, using games between the organisms for determining their fitness. Again, since this method required a very large amount of games, the method evolved only a few evaluation function parameters with limited success. This game playing approach facilitates the use of GA for efficiently evolving an evaluation function's parameters. As our results will demonstrate, this method is very fast, and the evolved program is on par with the world's strongest checkers programs. As already described, our goal is to evolve the parameters associated with board squares so that its evaluation function would produce as close a score as possible to the evaluation function of the same position.

VI. CONCLUSION

Comparative Result Analysis of Time Analysis and Weight Function parameters improvement using Graphical Representation has been shown in following Figure 2 and 3 where we can clearly visualize the effect of evolutionary genetic approach on all checkers evolving parameter.

One major problem with this program is that it takes a lot of time for a simulation to take place assuming one would like to simulate for at least 25 generations. For which the results were collected. A solution has taken a very small search depth, like two ply depth for both the opponents resulting in total depth of 4. The program was implemented using the search algorithm of Min-Max search with Alpha-Beta Pruning.

For many problems, evolutionary algorithms can often find good solutions (near-optimal) in around 50-100 generations. This can be many times faster than an exhaustive search. Here the Time Analysis chart, it is seen slow and incremental growth-decline pattern is observed in time requirement to play the game of Checkers for a specific sub-set of Generations. The collected results show very clear Evaluation Function improvement in each of the categories of Checkers discs positions. The range of the Minimum and Maximum Fitness parameters shows steady improvement for each set of subsequent Generations which shows the impact of Genetic Approach on them resulting in a better Evaluation Function learning.

REFERENCES

[1] R. Fortman, *Basic Checkers* (http://home.clara.net/davey/basicche.html, 2007).

[2] J. Schaeffer *et al.*, "Solving Checkers" (www.ijcai.org/papers/0515.pdf, 2005)

[3] Seo, Y.G., Cho, S.B., Yao, X.: Exploiting coalition in co-evolutionary learning. In:Proceedings of the 2000 Congress on Evolutionary Computation. Volume 2., IEEE Press (2000) 1268{1275

[4] Chellapilla, K., Fogel, D.: Evolving a neural network to play checkers without human expertise. In Baba, N., Jain, L., eds.: Computational Intelligence in Games. Volume 62. Springer Verlag, Berlin (2001) 39{56

[5] Fogel, D., Hays, T., Hahn, S., Quon, J.: A self-learning evolutionary chess program. Proceedings of the IEEE 92 (2004) 1947{1954

[6] Kendall, G., Whitwell, G.: An evolutionary approach for the tuning of a chess evaluation function using population dynamics. In: Proceedings of the 2001 Congress on Evolutionary Computation CEC2001, IEEE Press (2001) 995{1002

[7] Kusiak, M., Wal»edzik, K., Ma¶ndziuk, J.: Evolution of heuristics for give-away checkers. In Duch, W., et al., eds.: Arti¯cial Neural Networks: Formal Models and Their Applications - Proc. ICANN 2005, Part 2, Warszawa, Poland. Volume 3697 of LNCS.Springer (2005) 981{987

[8] Ma¶ndziuk, J., Osman, D.: Temporal di®erence approach to playing give-away checkers. In Rutkowski, L., et al., eds.: 7th Int. Conf. on Art. Intell. and Soft Comp.(ICAISC 2004), Zakopane, Poland. Volume 3070 of LNAI., Springer (2004) 909{914

[9] Osman, D., Ma¶ndziuk, J.: Comparison of tdleaf(,) and td(,) learning in game playing domain. In Pal, N.R., et al., eds.: 11th Int. Conf. on Neural Inf. Proc. (ICONIP 2004),Calcutta, India. Volume 3316 of LNCS., Springer (2004) 549{554

[10] Osman, D., Ma¶ndziuk, J.: TD-GAC: Machine Learning experiment with give-away checkers. In Drami¶nski, M., Grzegorzewski, P., Trojanowski, K., Zadro_zny, S., eds.: Issues in Intelligent Systems. Models and Techniques. Exit (2005) 131{145

[11] Pollack, J.B., Blair, A.D., Land, M.: Coevolution of a backgammon player. In Langton, C.G., Shimokara, K., eds.: Proceedings of the Fifth Arti¯cial Life Conference, MIT Press (1997) 92{98

[12] T. M. Mitchell. Generalization as search. *Artificial Intelligence,* 18:203 - 226, 1982.

[13] Paul Marcos Siqueira Bueno and Mario Jino. Identification of potentially infeasible program paths by monitoring the search for test data. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE)*, Grenoble, France, September 2000.

[14] S. Edelkamp, A. L. Lafuente, and S. Leue. Directed explicit model checking with hsf-spin. In *Proceedings of the 2001 SPIN Workshop*, volume 2057 of *Lecture Notes in Computer Science*, pages 57–79. Springer-Verlag, 2001.

[15] M.S. Campbell and T.A. Marsland. A comparison of minimax tree search algorithms. *Artificial Intelligence*, 20(4):347–367, 1983.

[16] 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.

[17] R. Gross, K. Albrecht, W. Kantschik, and W.Banzhaf. Evolving chess playing programs. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 740–747. Morgan Kaufmann Publishers, New York, 2002.