CHAPTER 4
OBSERVATION, RESULT AND DISCUSSION

4.1 INTRODUCTION

The idea of constructing computer programs modeled on the intellectual decision based on move making is motivational drive for systems which reveal acumen, wisdom aptitude and self-adaptation. The human brain has many exceedingly desired features that are hard to imitate in conventional computer systems. Incremental systematic efforts are made to acquire progressively sophisticated aptitudes over a span of definite time.

These machine learning based systems have shown the penchant of constantly altering and improving. Simultaneously it always preserves its truthfulness as a learning system. Such learning algorithms and methods are adaptive in nature and show elasticity to changes in its learning atmosphere, so that new practices and stimuli are amalgamated into evolutionary computing based system without altering existing competences. It displays to leverage computing power to withstand loss of intermediate results and the aptitude to self-evolve and self-reorganize in such a way that it retains developing functionality. [146]

Most of these techniques exploit proficient knowledge as main facet of learning as much as possible, such as the proper learning algorithm for training the evaluation function, feature centric relevance factors for the evaluation for board game and the weights evolution of the evaluation parameters. Acquiring such knowledge requires multidimensional assistance and assistance of game experts, computational power for
processing the knowledge mined, and a practice of trial and error to find the best overall approach.

4.2 THESIS APPROACH ..... 

So the most important question here is the "weights" evaluation function. If the weights are set correctly, board game computer program plays well. If they're not set correctly, the program may play very badly. It can try to use a "genetic" algorithm to learn good weights. The thesis approach think of the weights as "genes", and genetic operators "evolve" a good set of weight values. 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, the weights are kept in subsequent generation as survivor selection operator, and use them by making small changes using crossover onto them in the next generation version of the program.

But, just like with animals, evolution happens slowly over a large population of individuals likewise with programs. Instead of having just one possible move at a time, the game can have many. In this way it gets better and better individual weight sets each time. Let versions of the Evaluation function play against each other. It keeps the weight sets from the programs that win.

This evaluation regularly works just by calculating simple numerical features of the game position for example in Reversi, whether one player has more pieces, or is controlling the corners of the board. The final evaluation of the position is a number; it gets computing a linear function of the position features. That is, if the evaluation function is a chromosome string of length 10 depicting features of the board position, it calculates a number for each one, and then multiplies each numbers by its own "weight" value. This is because some of the features are more important, so we want to pay more attention to them.
4.3 GAME OF CHECKERS

4.3.1 Evolutionary Checkers

The thesis chooses Game of Checkers as its one of two board games as it relies on features that were very well chosen using human expertise and weighted by evaluation function tuning. It has its own dependency on volume of game moves’ for all possible exposable moves explored through extensive game board possibilities using min max search algorithm using alpha beta pruning. It is believed, once the best move exploration is reached, Computer playing Checkers will never make a slip-up because the final move outcome can be chosen from these states have already been evaluated. Game of Checkers aim to advance its play through better game rule understanding and perfect board positions information. It helps high-speed evolutionary computation to look ahead as many ply (search depth) as possible. The evolved evolutionary program exhibits a very stimulating flexibility that can be achieved with Checkers playing program generations until a highly tuned “fit” evaluation function gets emerged. By coding a given Checkers game playing problem algorithm into an Evolutionary computation framework, these clever algorithms are able to "evolve" solutions to real world problems.

4.3.2 Game Execution

The estimated quality of the board is calculated using the co efficient weights of the board squares to evaluate the leaf nodes of the tree Games. This interests considerable attention from AI researchers. The field of evolutionary algorithms is no exception to this rule. Over the years many games have been tackled with the evolutionary approach. A GA with genetic string signifying board game features and having evaluation weights as their co-efficient in evaluation function is been used for the game of Checkers playing program resulted in a competent player program that employed sophisticated mobility play.

4.3.3 Chromosome Structure
To determine the chromosome structure in this game tree, first phase is to divide checkers board into multiple sub-boards as per game experts understanding in sub-board patterns which helps in understanding the checker move and piece capturing logic and strategy. One 8x8 Checkers board can be divided into 36 3x3 sub-boards, 25 4x4 sub-boards, 16 5x5 sub-boards, 9 6x6 sub-boards, 4 7x7 sub-boards and 1 8x8 sub-board as shown in fig 4.1. It helps in dividing individual Checkers board piece position (which is 32) can be represented by it in genetic string.

Based on above depiction and understanding the genetic representation is initialized as a sequence of 32 genes, one per occupied board square as shown in Fig. 4.2. Each gene has basic status value considering +1 if occupied by player disc, -1 if it has opponent’s disc and 0 if it is empty. Each gene outcome is a fitness value. The gene value is decoded into a move by multiplying the basic value by the total number of moves based calculated evaluation value. This is based on “goodness” of board square which gets calculated on basis of game features’ significance, calculated min-max algorithm value and current status values available. This is used as an index for each single move from the list of available moves.
Observation Result and Discussion

Fig 4.2 Board Squares Numbers of Game of Checkers

For each possible move such values are calculated and gene giving the best possible move is moved to capture opponent’s position which increases game winning chances as it gives better placing of dics. Though this model with future moves dependent on the current move choice as the number of available moves at each node in the game-tree does vary significantly, it still serves to examine whether the co-evolutionary approach can produce useful game-play. [146]

Table 4.1 Checkers Evolutionary Process Algorithm

```
for each candidate in the population do
    Initialize candidate with random solution
    Evaluate each individual
end for
repeat
    EVALUATE population;
    RECOMBINE pairs of parents;
    MUTATE the resulting children;
    REINSERT new candidates;
    TERMINATE the non-parents;
until maximum fitness
```
After each move is made, the corresponding gene is cropped from all chromosomes structure of 32 genes, resulting in a chromosome structure that shortens as the game progresses. That particular chromosome gene or board piece is considered when opponent player considers that place to make a capturing move.

To optimize the move making process genetic operators are applied at the end of search algorithms. After applying min-max search and alpha beta pruning for ply depth of three at every single move making instance. The search process gives very high number of leaf nodes each one depicting one move at the depth of one, two and three. Now to attach the corresponding fitness value to each of this leaf node chromosome string based genetic optimization process is carried out as shown in table 4.1.

For the First population candidates are given random fitness values. Next, the loop is entered in which the best candidates are selected as parents. After that, the variation operators are applied. First, pairs of parents are recombined. The pieces of the candidates that are recombined are determined based on roulette wheel selection method. Next, single point mutation is applied in very small number. The parts that are mutated are chosen randomly. The next step in the loop is reinserting the new candidates (the children). Finally, the individuals that were not parents are terminated. This evolutionary process algorithm is shown in Table 4.1.

4.3.4 Fitness Function

Fitness function is the significant parameter of the genetic algorithm that defines the fitness of each chromosome where the values of genetic parameters are revised as the genetic evolution progresses. At ever generation, fitness value of each chromosome is calculated using fitness function. If fitness function 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 or local minima whichever is applicable. Once there is an improvement in the overall fitness, the original mutation rate is restored to continue evolution as normal.
The static board evaluation function is essentially a weighted-sum of features score based on the various properties of the board. The board features considered for the game of Checkers is given in Table 4.2. They are used to evaluate a terminal board properties marked by human expert players such as: number of pieces, mobility count, center-control, advancement of pieces, etc.

Thus evaluated score for a board may be viewed as simple linear polynomial, usually represented as follows:

\[ \text{Fitness Weight Value } F = (W_1 \times F_1) + (W_2 \times F_2) + \ldots + (W_n \times F_n) \] ----4.1

Where

The function value F is called the static Fitness Function to evaluate a game board configuration

The Fi's are functional features that play important roles in game-playing strategies

The Wi's are weights that indicate the occupancy of the specific board position.

<table>
<thead>
<tr>
<th>Table 4.2 Functional Features for Game of Checkers</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Number of discs (Total Disc count) = f₁</td>
</tr>
<tr>
<td>• Number of playable squares (aka mobility) = f₂</td>
</tr>
<tr>
<td>• Potential capture places (aka frontier. That is the number of empty squares diagonal to opponent stones) = f₃</td>
</tr>
<tr>
<td>• Parity (Who will put the last stone if no pass occurs) = f₄</td>
</tr>
<tr>
<td>• Disc pattern = f₅</td>
</tr>
<tr>
<td>• Existence of any other patterns. = f₆</td>
</tr>
</tbody>
</table>

Every major feature value Fᵢ (where i= 1, 2, 3,…n) is the resultant value of the above mentioned pattern features and each one is having fixed and pre-determined values given to them. Thus values of F₁, F₂ ……Fn can be calculated as follows:

\[ F_1 = f_{11} + f_{21} + f_{31} + f_{41} + f_{51} + f_{61} \]
\[ F_2 = f_{12} + f_{22} + f_{32} + f_{42} + f_{52} + f_{62} \]
\[ \ldots \]
Observation Result and Discussion

\[ F_n = f_{1n} + f_{2n} + f_{3n} + f_{4n} + f_{5n} + f_{6n} \]

This results in \( F_i = \sum f_{ij} \) where \( i = 1, 2, \ldots, 6 \) and \( j = 1, 2, 3, \ldots, n \)

It is noted that for certain disc positions in some fixed situations the value of \( f_{ij} \) can be zero also if total disc count has no role to play in terms of that specific board position analysis. As explained the values are fixed and pre-determined based on game of Checkers expertise and game move analysis. The importance of each feature is fixed and having equal weightage among them. The contributing values are different and selected from pool set of fixed values based on the board position it is referring to.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>…</th>
<th>31</th>
<th>32</th>
</tr>
</thead>
</table>

Fig 4.3 Checkers Chromosome string based on Board Piece Position

### 4.4 CHECKERS PROGRAM IMPLEMENTATION

In an evolutionary genetic algorithmic style, each checkerboard can be represented by a genetic vector with length of 32 as shown in Fig. 4.3 with each vector representing one available position on the board. As the evaluation function is in a linear form here vector components are elements having the values −1, 0, +1, where 0 corresponded to an empty square, 1 was the value of a regular checker, and −1 was the number assigned for an opponent’s checker piece. +k represent player’s king and −k represent opponent’s king. Here the sign of the value for piece was suggestive for the piece belongingness, either to the player (positive) or his/her opponent (negative).

Any move from a player’s is determined by evaluated fitness function parameters of the board game positions multiplied by their related vector weight values. Thus evaluation function is formulated for the evolutionary landscape parameter values. The first set of vector values is randomly set using for the first generation experienced human expertise. These random set of values are designed to indicate the spatial characteristics of the game board. Then after the remaining subsequent values are computed that are based on evolutionary computation process. It primarily evolves the genetic weight parameters.
These weight values and the functional features’ values are in dot product formation gives final fitness values which forms the basis for the move selection. These values become base for search algorithm like min-max and the better value is chosen to make the next move. Above procedure is repeated for a reasonable ply depth of three. Here it is noteworthy that more depth selection makes the algorithm slower as exploration of all possibilities for all depth values take a lot of time. It hampers the evolutionary benefits of better move selection in reasonable time.

In an initial board set up the complexity enhancement is because of board weights and their respective holdings which are worth of the board squares importance (based on its substantial features) and its relative importance in game tree. If these findings are favorable to the player’s interest then these pieces take positive values. If very initially the potential move itself is set closer to the final output then it takes value 1.0, which denotes that corresponding board square is better. But if the move takes potential output to an opponent’s favor the square is set to –1.0, which means the board square situation is worsening which may result in combined loss over a long run of moves. All budding 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.

In a genetic evolutionary process each “parent” is generated as an off springs by changing all of the associated weights and related square values. Here all parents and their offspring compete for survival by playing games moves of checkers and receiving weights as the results of their play. Each player scored –1, 0, or +1 points for a loss, draw or win, respectively. These figures are stand-alone and contain no optimality parameter into it. A draw is declared in case if a game lasts for 100 moves. In total, each game has an average of 40-60 moves and such 50 games are been played. In each generation within a game, each checker piece is observed for its participation and corresponding fitness values it possesses. After all move possibilities are explored, all the participating checkers squares that received the greatest total fitness weight points across all fitness value analysis and computation are reserved as parents for the next generation and the
evolutionary genetic process gets repeated for considerable number of time to accomplish better fitness results.

Each game tree is explored by applying a min-max alpha-beta search algorithm with a look ahead over a selected number of moves. The ply depth of the search, $d$, was set at three (i.e., three depth move travelling for each side) to allow for reasonable execution times (50 generations). The depth ply is extended in units of three any time a forced jump move was encountered because in these situations no real decision is available.

The every move making place the best move is selected on values collected over running min-max repeatedly for all possibility in a given depth. These moves positive and negative depth are dependent on the side of the player. For the player the weight notion is positive and for opponent the weight sign is set to negative.

4.5 CHECKERS COLLECTED RESULTS

Total 50 Checkers games against human players are been 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 $d = 3$ to ensure reasonably mature depth exploration and thus limiting only 50 games as the used to consume a reasonably good amount of time.

With a population size was set to 50 and 100 and the process was run for a set of 25, 50 and 100 generations with genetic operator parameters selected as shown in Table 4.3.

<table>
<thead>
<tr>
<th>Table 4.3 Genetic Operators for Game of Checkers</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Selection rate = 0.20</td>
</tr>
<tr>
<td>• Crossover rate = 0.80</td>
</tr>
<tr>
<td>• mutation rate = 0.01 (occasional)</td>
</tr>
</tbody>
</table>

The Game of Checkers implemented in Java is run for above mentioned parameters and various screen shots, fitness value graphs are shown in following screen shots and graphs as shown below.
The following screen shots are showcasing the perfect flow of evolutionary checkers game implementation and the functional features of the Board game. The figures are showing the different levels of evolutionary Checkers and the initial board game arrangement.

It shows the move making and capturing moves by different moves. It not only highlights the Checkers program capability to make a piece King by going into specific board position row but also highlights the feature that King checker piece can move backward and capture opponent’s piece in last figure.

It is proven that Evolutionary Checkers implementation is complete as per the game rules and it shows positive learning when computer program make moves of capturing sing various functional features like move pattern, mobility, parity, disc pattern and identification of higher return giving capture position analysis and move making.

Fig 4.4 Checkers Program showing Levels of Game Play
Observation Result and Discussion

Fig 4.5 Checkers Program showing Initial Board Configuration

Fig 4.6 Black making first move
Observation Result and Discussion

Fig 4.7 Showing possible Capture for Red Disc

Fig 4.8 Showing captured Black turning into Red disc
Fig 4.9 Depicting Mobility for Black Disc

Fig 4.10 Red Disc turning into King
Fig. 4.11 King making backward move (Capture)

Fig. 4.12 King after making backward move (Capture)
Following are figures which show various results of fitness weight values collected for various game program generations. The graphs are representing the evolutionary weight changes evolved over a span of various generations.

Figure 4.13 and 4.14 show the evolution of Max. fitness value changes for Checkers board positions number 14 and 08. For fifty generations the values are collected, initial drop in fitness values show the gradual machine learning process which depicts the move learning. After twenty generations the values are of higher numbers and some crossover results show occasional drop in maximum fitness values. For checker board positions 10, 18, 20 and 26 the fitness value graph is drawn spanning fifty generations showing the minimum and maximum fitness values. The first two board positions are more easy to capture and thus show very turbulent fitness value changes.

Fig 4.13 Max. Fitness values for Board position 14

Fig 4.14 Max. Fitness values for Checker Board position 08
Fig 4.15  Max. and Min. Fitness values for Checker Board position 10

This show their mobility index as well as capture possibilities. While other checker board positions 20 are more stable and offers very steady and stable fitness values. Board position  26 is highly unwavering and has a very smooth capture prospect. It shows smooth surge in maximum fitness values and steady minimum fitness progress in fifty generations.

Fig 4.16  Max. and Min. Fitness values for Checker Board position 18
Observation Result and Discussion

The results of Checkers’ board position 9, 10, 11 and 12 as their positions shown in fig 4.19 are in straight row and stable one. After initial drop in fitness values for a set of game generations the values are almost stable and identical for each of these generations. Some piece captures happening in these rows show some rise in Fitness values as capture gives better chances of strong move in some generations of game. Figure 4.20 shows identical graph for Checkers’ board positions 21, 22, 23 and 24. In the graph board position 21 and 24 shows more piece mobility and positive capture incremental value rise.
Fig 4.19  Max. and Min. Fitness values for Checker Board position 9,10,12 and 12

Following two figures 4.21 and 4.22 shows various fitness values (maximum and minimum) attained by a Checker piece which eventually turned into King. The graphs are made for two kings of Red and Black color for different number of moves. The final points which show steep raise in their fitness values which clearly show the checker piece turning into King.
Observation Result and Discussion

Fig 4.21 Fitness Scores attained by Red Checker King for different moves

Fig 4.22 Fitness Scores attained by Black Checker King for different moves

Following figure 4.23 shows the fitness chart accomplished by Red and Black checkers piece for a span of 100 generations for maximum fitness values they have gained. The graph clearly show that for these hundred generations Red piece have got more fitness values than Black pieces. From the graph it can be inferred that Red has attained better capturing positions and Red checker side final wins the game.
Observation Result and Discussion

4.6 GAME OF REVERSİ

The board of the Reversi is 8x8, and each lattice can be black, white or empty. There are $3^{64}$ (about $3.4 \times 10^{30}$) possible states, even if it is considered the symmetric and erase part of the illegal states (which are not possible to appear); there are still a huge number of states which require an extremely high space complexity to store them. Therefore, before the winning strategy has been proved, the computer Reversi still have room to be improved.

In the rule of the Reversi, it can turn over the adversary’s pieces only after a legal move. For each hand, there are around 1 (including pass) to 15 (rarely more than 15) legal moves which is regular of eight moves. It is perceived that Reversi’s possible moves for each hand is much fewer than other board games compare to game of Go game which has more than 200 possible moves for each hand or Chinese Chess which has more than 50 possible moves for each hand. Even if the designing of the evaluation function are not accurately, with the help of high speed calculation ability of the recent computer, we may still get a fair result after several layers of min-max search and alpha-beta pruning when design the computer Reversi program.

![Fitness Chart of Red & Black Disc](image)

Fig 4.23 Fitness Scores attained by Black Checker King for different moves
This thesis grants a more “human-like” approach to game playing by evolving artificial game playing by taking genetic approach. Before developing the program for computer to play the game of Reversi it is very well observed that the disc are black on one side and white on other side. It has same value so far as their individual entity score is concerned in contrast to game of Chess where each piece has different value and moving pattern.

In Reversi it is the board position which is important in terms of placing the disc and flipping pattern of opponent disc by bracketing it in direction of horizontal, vertical or diagonal. For example corner discs positions denoted by H in figure 4.25 once captured can never be revert back by any opponent and that is the reason they are called stable discs with higher degree of weightage by any player who wants to increase its disc count in any phase of the game.

Discs like C, E and E are having more stability than disc like K, M, J, I and X which can be easily captured. But K, M, J, I and X provides better mobility than C, E and F. So stability and mobility are two important features for the game of Reversi and they are part of features considered to make the Evaluation function for the game player of the thesis.
From the above figure it is observed and made evident that to develop very skillful program in move making and reducing the huge search space the Reversi board of sixty-four squares is divided in a set of ten families (Fig. 4.26).

![Reversi Board Discs Set](image)

Here each set of board squares was assigned a multiplying coefficient based on the importance of that particular set as been discussed in previous paragraph under the feature analysis of mobility and stability.

This group coefficient of ten functions sets underwent coefficient change as genetic operators were applied on them. The ten disc set is the basic and main chromosome string as shown in fig 4.26 to apply the genetic operators like selection, crossover and a small degree of mutation onto it. So min max and alpha-beta search algorithm searches various possible moves and once such leaf nodes are available based on their board positions then after to evaluate their fitness values to decide the move making decisions this chromosome string is used to find the fitness values by applying genetic operators onto them.

![Reversi Chromosome string based on Ten Disc Family](image)
4.6.1 Fitness Program Implementation

For any board game constructing an Evaluation function is very important and critical. The building of evaluation function usually works just by calculating simple mathematical features of the game position (In Reversi Game, aim is to put the player is better place with respect to stability and mobility.) The Reversi evaluation function is a number; which is got by computing a linear function for the positional features. Positional features are based on how the evaluation function uses six features of the position, it calculates a number for each one, and then multiplies each numbers by its own "weight" value.

The board is represented as a vector of length 64. Black disc is represented as 1 and white disc as -1. Empty space is 0. The relevance of the board is calculated using weighted piece counter. The weighted piece counter is a vector of length 64. Each element corresponds to one square in the Othello board, which means the weights of the board position. The cumulative fitness value of the board squares is calculated using dot product of two vectors (weights and functional features) as follows:

$$\text{Fitness Weight Value } F = (W_1 \times F_1) + (W_2 \times F_2) + \ldots + (W_n \times F_n) \quad \ldots 4.2$$

Where

The function value $F$ is called the static evaluation of a game board configuration

The $F_i$'s are functional features that play important roles in game-playing strategies

The $W_i$'s are weights that indicate the relative importance of the features.

The weights of each position are initialized as a value between -1 to 1. Until out-of-opening, the weightings in the population use opening knowledge. After out-of-opening, the weightings are evaluated in the relevance of board and the next move is decided. Among possible moves, the one with the highest fitness weight value ($f$) is selected. For game of Reversi the Evaluation function game feature metrics constituents is shown in table 4.4.
Table 4.4 Reversi features for Fitness function formation

- Number of stable discs (aka stable disc count) = \(f_1\)
- Number of playable squares (aka mobility) = \(f_2\)
- Potential mobility (means frontier that is the number of empty squares aside opponent stones) = \(f_3\)
- Parity (Who will put the last stone if no pass occurs) = \(f_4\)
- Edge pattern (pre-computed evaluation for each 10 squares) = \(f_5\)
- Corner pattern (10 squares on a triangle pined at corner plus one more diagonal square as shown in Figure 4.2) = \(f_6\)

Every major feature value \(F_i\) (where \(i = 1, 2, 3, \ldots n\)) is the resultant value of the above mentioned pattern features and each one is having fixed and pre-determined values given to them. Thus values of \(F1, F2 \ldots Fn\) can be calculated as follows:

\[
F_1 = f_{11} + f_{21} + f_{31} + f_{41} + f_{51} + f_{61}
\]
\[
F_2 = f_{12} + f_{22} + f_{32} + f_{42} + f_{52} + f_{62}
\]
\[
\ldots\ldots
diagonal
\]
\[
F_n = f_{1n} + f_{2n} + f_{3n} + f_{4n} + f_{5n} + f_{6n}
\]

This results in \(F_i = \sum f_{ij}\) where \(i = 1, 2, \ldots 6\) and \(j = 1,2,3,\ldots n\)

Like game of Checkers, in game of Reversi also it is noted that for certain disc positions in some fixed situations the value of \(f_{1j}\) can be zero also if total disc count has no role to play in terms of that specific board position analysis. As explained the values are fixed and pre-determined based on game of Reversi proficiency and game move study. The importance of each feature is fixed and having equal weightage among them. The contributing values are different and selected from pool set of fixed values based on the board position it is referring to.

The end search evaluation function majorly focuses on the usage of disc patterns which has small edge or parity feature associated with it. All these components are incrementally restructured to achieve a good depth of search. The collective fitness value of all the board squares is calculated using dot product of two vectors (weights
represented by $W_i$ which are disc weight and functional features shown by $F_i$ which are collective feature values of all feature metrics constituents mentioned in previous paragraph) as follows in equation 4.1. Here the function value $F$ is called the static evaluation of a game board configuration and is the fitness function value for a given board configuration.

For the opening set of weights of each position are initialized as a value between -1 to 1 as their opening values. These opening values are used for the first set of generation. After the first pass gets over, the weight values are evaluated as per the new discs positioning of the board which helps in choosing-deciding the next move. For each of the possible move the fitness weight value $F$ is calculated for each of the possible depth values.

After all “$F$” results are calculated for all possible positions. To make the next move selection, the one with the highest fitness weight value ($F$) is selected. After the move has been made for the next move the whole above mentioned process for the all possible move for all depth variant types are explored and calculated and this loop is repeated for the entire game of Reversi to find the winner of the game.

In above case, the chromosomes (potential solutions) are expressed as fixed length binary strings. Reversi has a binary string of 10 bits which are based on symmetry feature of the board game. These ten discs set gets simulated throughout the Reversi board so their relative weight for each of the disc set is same across board. It is also known that any fitness function defined on binary strings can be evaluated by assigning fitness values to them and subsequently gets evolved over a span of defined set of generations.

The evaluation function proposes that output of the quality of each possible move at the current board configuration. The function along with min-max search in the game of Reversi at each move making level saw the updated board and gives the rank of each move and only a subset of these moves was explored in limited ply depth. The function
identified several important principles of game play and used them as the basis for
 genetic evolving of populations.

### 4.7 REVERSI RESULTS

The Game of Reversi program is performed on a Pentium machine with the RAM size of
1 GB. The results are collected for two discs sets. The population size was set to 100.
Evolutionary weight behavior was visible even with this relatively small population size.
The program employed fitness-proportional selection method. Game of Reversi relies on
expertize captured in an evaluation function, which is used to find out next move by
exploring and evaluating current possible moves in the stage. Reversi’s success is based
largely on traditional game theory mechanics (min-max game tree and alpha-beta search)
and genetic optimization cycle. The program implementation algorithm is shown in Table
4.5.

In game implementation the program is iterated for a considerable amount of generations
with a ply depth of one, two and three. The program took a considerably reasonable
simulation time for a simulation time for a handful set of generations of fifty. For the
Reversi program, the genetic operators are being used in each population formation of all
generations and shown in table 4.1.

#### Table 4.5 Reversi Implementation Algorithm

1. Start with a randomly generated population of n
   chromosomes, all of fixed length. (10 bit string chromosome)
2. Calculate the fitness $F(c)$ of each chromosome $c$ in the population.
   If a solution has been found, i.e. $F(c) = T$ for some c, and predefined target T (Target T is
   Win in Reversi Game or no dics is empty condition), then stop. Otherwise,
4. Repeat the following steps until n offspring have been created:
   a. Select some pair of chromosomes as parents for the
      creation of offspring. (Selection)
   b. Recombine the genetic material from the two parents to form two offspring
      (Recombination)
   c. Mutate p randomly selected genes on each of q randomly selected offspring
      (Mutation).
5. Replace all the chromosomes in the current population with the new offspring.
6. Go to step 2 for next Generation
The Reversi is run for 25, 50 and 100 generations for population sizes of 50 and 100. Mutation as it flips the genetic string value is set very low because of its impact on the fitness of mutated string.

In the experiments conducted in this thesis, the ‘fitness proportionate’ selection method has chosen for pairing parents. Out of two traditional recombination operators: the ‘single point crossover’ and the ‘uniform crossover’, uniform crossover is used as crossover method keeping the disc family set representing disc position for each type in mind.

<table>
<thead>
<tr>
<th>Genetic Operator</th>
<th>Size</th>
<th>Size</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Generations</td>
<td>50</td>
<td>100</td>
<td>25</td>
</tr>
<tr>
<td>Population Size</td>
<td>100</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Survivor</td>
<td>10%</td>
<td>20%</td>
<td>10%</td>
</tr>
<tr>
<td>Crossover</td>
<td>90%</td>
<td>80%</td>
<td>90%</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>

After two computer players have played a game, the loser is replaced with a deterministic alternative of the winner by varying the weights for other set of discs or the determining features that were used, or by substituting features that had very low weight coefficient compared to other features.

4.7.1 Power Rate

The Reversi game is based on capturing opponent’s discs by making ‘smart’ and ‘strategic’ moves. In last player having maximum discs on game board wins the game. So apart from Fitness function there is another parameter which is based on the count of number of discs or number of wins to check the actual performance of evolutionary game of Reversi. That parameter is named
**POWER Rate** = Performance of Weight values for Evolutionary Reversi

The basic definition of Power rate is one parameter of comparing Evaluate Function. Power rate definition means percentage of pieces from total number of pieces (from all games) or percentage of wins. The equation to define Power Rate is as follows:

\[
\text{Power rate} = 100.0 \times \frac{\text{Pieces of player 1}}{\text{Pieces of player 1} + \text{Pieces of player 2}}
\]

\[
\text{OR}
\]

\[
\text{Power rate} = \frac{(\text{player 1 Wins} + 0.5 \times \text{draws}) \times 100}{\text{player 1 Wins} + \text{player 2 Wins} + \text{draws}}
\]

So Reversi results are collected as Generations results and stored as Generations (Gen 00, Gen 01, Gen 02 and so on) are always Text files which are converted into word files (.doc extension). This file stores the entire results/data of the simulation at each generation, enabling one to make a thorough analysis afterward.

The Evolutionary Game of Reversi implemented in Java is run for above mentioned parameters shown in Table 4.11 and various screen shots, fitness value graphs are collected and analyzed which are shown in following screen shots and figures as shown below.

The screen shots basically throws light at the overall flow of one full game playing and shows various aspects of Reversi game features like mobility, initial configuration of the game, importance of stable discs, parity means who places last disc on game board, different phases of the game like opening phase, middle stage of the game and final phase of the game.

The evolutionary Reversi program gives following play options:

- **Player against computer, player starts.** This option enables one to start the game. Only this time, the player makes the first move.
- **Player against computer, computer starts**: This option enables one to start the game. Only this time, the Computer makes the first move.
- **Two Human players**: This option enables one human player to play against another human, if he is too disgusted by playing against the computer.

![Game of Reversi – Welcome Screen](image1)

![Game of Reversi – Possible Play Option](image2)
Fig 4.29 Game of Reversi – Initial Board Configuration (Black to move first)

Fig 4.30 Game of Reversi – Black Makes first move
Observation Result and Discussion

Fig 4.31 Black and white both trying to capture more stable discs

Fig 4.32 Black capturing first stable corners
Fig 4.33 Black capturing both stable corners

Fig 4.34 Black building strong middle phase game through mobility
Fig 4.35 Black enriching on potential mobility

Fig 4.36 Black and White – Capitalizing on End phase moves
Fig 4.37 White capturing horizontal, vertical and diagonal discs

Fig 4.38 White making last move
The first result is the explanation of one generation and how genetic operators like survivor, crossover takes place between different genetic string and how second generation is formed by applying the operators onto them. It shows which two strings is crossover also.

The table clearly shows the formation of next generation of 100 population members by applying survivor 10% and cross over 90%. Here the fitness based selection for cross over is based on roulette wheel selection among the population members.

It clearly gives the idea of how the generation progresses and keeps the fitter individuals to the next generation which ultimately results in fitter population members. It helps in getting very result providing move decision and thus higher chances of wins are secured.
Table 4.7 Various Genetic operators to form next generation of strings

<table>
<thead>
<tr>
<th>Survivors:</th>
<th>Crossovers:</th>
<th>72 &amp; 25 -&gt; 45</th>
<th>9 &amp; 69 -&gt; 80</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 -&gt; 1</td>
<td>22 &amp; 23 -&gt; 11</td>
<td>72 &amp; 25 -&gt; 46</td>
<td>28 &amp; 49 -&gt; 81</td>
</tr>
<tr>
<td>12 -&gt; 2</td>
<td>22 &amp; 23 -&gt; 12</td>
<td>56 &amp; 5 -&gt; 47</td>
<td>28 &amp; 49 -&gt; 82</td>
</tr>
<tr>
<td>17 -&gt; 3</td>
<td>34 &amp; 56 -&gt; 13</td>
<td>56 &amp; 5 -&gt; 48</td>
<td>1 &amp; 60 -&gt; 83</td>
</tr>
<tr>
<td>21 -&gt; 4</td>
<td>34 &amp; 56 -&gt; 14</td>
<td>24 &amp; 93 -&gt; 49</td>
<td>1 &amp; 60 -&gt; 84</td>
</tr>
<tr>
<td>85 -&gt; 5</td>
<td>30 &amp; 60 -&gt; 15</td>
<td>24 &amp; 93 -&gt; 50</td>
<td>25 &amp; 89 -&gt; 85</td>
</tr>
<tr>
<td>6 -&gt; 6</td>
<td>30 &amp; 60 -&gt; 16</td>
<td>8 &amp; 94 -&gt; 51</td>
<td>25 &amp; 89 -&gt; 86</td>
</tr>
<tr>
<td>40 -&gt; 7</td>
<td>93 &amp; 70 -&gt; 17</td>
<td>8 &amp; 94 -&gt; 52</td>
<td>19 &amp; 42 -&gt; 87</td>
</tr>
<tr>
<td>10 -&gt; 8</td>
<td>31 &amp; 83 -&gt; 19</td>
<td>71 &amp; 20 -&gt; 53</td>
<td>19 &amp; 42 -&gt; 88</td>
</tr>
<tr>
<td>41 -&gt; 9</td>
<td>31 &amp; 83 -&gt; 20</td>
<td>58 &amp; 84 -&gt; 55</td>
<td>57 &amp; 30 -&gt; 89</td>
</tr>
<tr>
<td>75 -&gt; 10</td>
<td>42 &amp; 39 -&gt; 21</td>
<td>58 &amp; 84 -&gt; 56</td>
<td>70 &amp; 42 -&gt; 91</td>
</tr>
<tr>
<td></td>
<td>42 &amp; 39 -&gt; 22</td>
<td>7 &amp; 97 -&gt; 57</td>
<td>70 &amp; 42 -&gt; 92</td>
</tr>
<tr>
<td></td>
<td>82 &amp; 3 -&gt; 23</td>
<td>7 &amp; 97 -&gt; 58</td>
<td>90 &amp; 33 -&gt; 93</td>
</tr>
<tr>
<td></td>
<td>82 &amp; 3 -&gt; 24</td>
<td>2 &amp; 80 -&gt; 59</td>
<td>90 &amp; 33 -&gt; 94</td>
</tr>
<tr>
<td></td>
<td>60 &amp; 99 -&gt; 25</td>
<td>2 &amp; 80 -&gt; 60</td>
<td>15 &amp; 5 -&gt; 95</td>
</tr>
<tr>
<td></td>
<td>60 &amp; 99 -&gt; 26</td>
<td>78 &amp; 94 -&gt; 61</td>
<td>15 &amp; 5 -&gt; 96</td>
</tr>
<tr>
<td></td>
<td>14 &amp; 56 -&gt; 27</td>
<td>78 &amp; 94 -&gt; 62</td>
<td>87 &amp; 22 -&gt; 97</td>
</tr>
<tr>
<td></td>
<td>14 &amp; 56 -&gt; 28</td>
<td>23 &amp; 57 -&gt; 63</td>
<td>87 &amp; 22 -&gt; 98</td>
</tr>
<tr>
<td></td>
<td>96 &amp; 29 -&gt; 29</td>
<td>23 &amp; 57 -&gt; 64</td>
<td>54 &amp; 89 -&gt; 99</td>
</tr>
<tr>
<td></td>
<td>96 &amp; 29 -&gt; 30</td>
<td>23 &amp; 19 -&gt; 65</td>
<td>54 &amp; 89 -&gt; 99</td>
</tr>
<tr>
<td></td>
<td>15 &amp; 93 -&gt; 31</td>
<td>23 &amp; 19 -&gt; 66</td>
<td>54 &amp; 89 -&gt; 100</td>
</tr>
<tr>
<td></td>
<td>15 &amp; 93 -&gt; 32</td>
<td>93 &amp; 52 -&gt; 67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>89 &amp; 29 -&gt; 33</td>
<td>93 &amp; 52 -&gt; 68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>89 &amp; 29 -&gt; 34</td>
<td>23 &amp; 54 -&gt; 69</td>
<td></td>
</tr>
<tr>
<td></td>
<td>96 &amp; 70 -&gt; 35</td>
<td>23 &amp; 54 -&gt; 70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>96 &amp; 70 -&gt; 36</td>
<td>71 &amp; 9 -&gt; 71</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 &amp; 38 -&gt; 37</td>
<td>71 &amp; 9 -&gt; 72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 &amp; 38 -&gt; 38</td>
<td>68 &amp; 37 -&gt; 73</td>
<td></td>
</tr>
<tr>
<td></td>
<td>86 &amp; 65 -&gt; 39</td>
<td>68 &amp; 37 -&gt; 74</td>
<td></td>
</tr>
<tr>
<td></td>
<td>86 &amp; 65 -&gt; 40</td>
<td>25 &amp; 80 -&gt; 75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 &amp; 92 -&gt; 41</td>
<td>25 &amp; 80 -&gt; 76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 &amp; 92 -&gt; 42</td>
<td>93 &amp; 65 -&gt; 77</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 &amp; 19 -&gt; 43</td>
<td>93 &amp; 65 -&gt; 78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 &amp; 19 -&gt; 44</td>
<td>9 &amp; 69 -&gt; 79</td>
<td></td>
</tr>
</tbody>
</table>
Reversi program implementation is based on two novel ideas. The first one is to use genetic string where individual fitness of board square is improved by applying linear fitness function based genetic operators. The second is to use the symmetry feature of the game as basis to divide the sixty four board squares into ten basic families. This not only reduces the chromosome string and min- max search process but helps in finding the fitness weight improvisation in a better way which are been collected over a varying span of generation numbers and population sizes as shown in table 4.7.

Following are figures which show various results of fitness weight values collected for various game program generations. The graphs are representing the evolutionary weight changes evolved over a span of various generations.

First two figures show fitness analysis of twenty five and fifty generations respectively for minimum and maximum fitness values. The first graph shows considerable rise in min. fitness over generations and sudden fall into it shows the impact of mutation results. Second graph shows steady rise in the collected values for these parameters.

![Fitness Analysis Graph](image)

**Fig 4.40 Fitness Analysis of 25 Generations for Maximum and Minimum Fitness**

Next two graphs learns the impact of evolutionary fitness value increment in successive generations of one, five and ten for figure 4.42 and generations of 20 and 30 for figure 4.43. First graph has for rapid value changes which shows initial machine learning. Second graph is more stable in move making selection process and shows balanced rise in fitness values.
Observation Result and Discussion

Fig 4.41 Fitness Analysis of 50 Generations for Maximum and Minimum Fitness

It shows the very clear learning pattern of the all disc position to build up good moves and fitness progress which is very much required to exhibit good move making and winning play in later generations of the game. The graph shows that genetic progress over the span of 20 generations.

Fig 4.42 Fitness Analysis of Three Successive Generations for Population size 100

As stated earlier that the program mainly concentrates on evolutionary fitness analysis of ten disc families’ next four graphs highlights the fitness weight values collected over a span of fifty or hundred generations. Corner disc positions are most stable in Reversi board and their fitness values remains almost stable once they are captured. The steep drop shows particular disc captured by opponent.
Fig 4.43 Fitness Progress Analysis of two Generations spanning 10 generations

Fig 4.44 Weight Change Analysis of Corner (Stable) Disc Position

Fig 4.45 Weight Change Analysis of Adjacent to Corner (C1 and C2) Disc Position
Adjacent to corner disc are less stable and thus has some shaky fitness values movement in different generations. Such two positions of C are depicted by C1 and C2 in figure 4.45. Fitness charts of board position X and Y are shown in figure 4.46 and 4.47. Both these positions are very participative in complete span of game and thus values in subsequent generations show moderate degree of value changes but these values are of higher fitness magnitude and remains in that window which is the objective of evolutionary machine learning.

Next two figures (4.48 and 4.49) shows fitness value fluctuations in initial board configurations of white discs and black discs separately specific set of moves. Collected
results show high fitness alterations (positive and negative values) as the discs can be held by player as well as opponent also. These board positions has vital role in initial phase of the game and flips easily as it’s positions falls in the middle of the game board.

Fig 4.48 Fitness Chart of various values by White Pieces on Initial Board Position

Fig 4.49 Fitness Chart of various values by Black Pieces on Initial Board Position

Fig 4.50 Fitness Chart of various values by Board Position J and K
Figure 4.50 shows fitness value chart for board position J and K. Position K has major value changes than position J for these given span of one hundred generations. These Fitness values are very promising to note that the game program learns the move making process using genetic operators.

Power rate equation shown in equation 4.3 shows the overall performance of computer player program playing the game of Reversi. The graph shows steady incremental performance improvement. The initial slow rise depicts the machine learning process and later on the curve gets good rise which is clear game play betterment.

Thesis results collected for both board games are very stimulating and progressive. They are analyzed for various parameters and their fitness function values evolved over a span of set of generations to give finer fitness results.

## 4.8 IMPLEMENTATION COMPARISON WITH EARLIER APPROACHES

The thesis makes a novel approach to implement the board game realization of Checkers and Reversi. It has employed evolutionary approach which are feature based genetic string formation and evolution of these weight values over a period of generations with
Observation Result and Discussion

varying sizes of population. This process lasts for specific and reasonably moderate number of games. The thesis shows comparison of above mentioned approach along with collected results to earlier implementation of Game of Checkers and Game of Reversi.

4.8.1 Implementation Comparison - Game of Reversi

Since many decades researchers and program developers implemented Reversi by adopting various learning approaches. Some of them are discussed as follows along with their tabular comparison. Game of Reversi program BILL2.0 has achieved remarkable success at the world championship level. It was based on two stages: training and recognition. In training stage, a database of branded training positions were collected from real games and named as winning positions and losing positions. The program has an issue of mislabeling, e.g., that Bill lost from a position in which an optimal player would win.

BILL2.0 was also based on a linear evaluation function that gets tuned carefully. It was generating 20 arbitrary moves in initial phase of game and then each side played the residual 40 half-moves in 15 minutes and the last 15 moves were played using perfect endgame search. An entire of 3000 games were played and their positions were recorded as training data. To acquire the training positions, two replicas of BILL 2.0 are used to play with each other from early positions.

Various Reversi program implementations along with their features are been compared with the thesis implementation is elaborated as shown in table 4.8.

4.8.2 Implementation Comparison –Game of Checkers

Some of the earlier approaches to implement Game of Checkers program can be describes as follows:
Table 4.8 Reversi program comparison

<table>
<thead>
<tr>
<th>Game Program Parameter</th>
<th>Name of Reversi Program</th>
<th>IAGO</th>
<th>Bill</th>
<th>Logistello</th>
<th>Thesis Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search Method</td>
<td>Alpha-Beta Search Algo.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Improved search techniques</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Selective Search + PRoBCuT</td>
<td></td>
<td></td>
<td></td>
<td>Min-Max Search+ Alpha-Beta Pruning</td>
</tr>
<tr>
<td>Feature used</td>
<td>Stability &amp; Mobility</td>
<td></td>
<td></td>
<td>Four (Mobility+ Pattern+ Parity+ Stability)</td>
<td>Six features (Stability+ Mobility+ Disc Count+ Parity+ Corner pattern + Specific Pattern)</td>
</tr>
<tr>
<td></td>
<td>Extended set of Features using pre computed tables</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eval. Function based</td>
<td>Single + Linear</td>
<td></td>
<td></td>
<td>Simple</td>
<td>Single Linear Evaluation Fun</td>
</tr>
<tr>
<td>No. of Eval. Fun Used</td>
<td>01</td>
<td></td>
<td>Several</td>
<td>Multiple</td>
<td>01</td>
</tr>
<tr>
<td>Game Database</td>
<td>--</td>
<td>3000 games</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branching Factor</td>
<td>5</td>
<td></td>
<td>--</td>
<td>--</td>
<td>Normal</td>
</tr>
<tr>
<td>No. of moves</td>
<td>64</td>
<td></td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>Training Based</td>
<td>No</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Specific Program Trait</td>
<td>Alpha-Beta + Iterative Deepening + Move Ordering</td>
<td></td>
<td></td>
<td>Bayesian learning + timing algorithm</td>
<td>Local mobility features+ table based evaluation scheme</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Genetic Operators on String of 10 disc family set</td>
</tr>
<tr>
<td>Developed by</td>
<td>Rosenbloom</td>
<td></td>
<td></td>
<td>K.F. Lee and S. Mahajan</td>
<td>Lee et el.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D Singh, Shah and C S Thaker</td>
</tr>
<tr>
<td>Specific Remark</td>
<td>--</td>
<td>Bill &amp; Bill 2.0</td>
<td></td>
<td></td>
<td>PoWER Rate</td>
</tr>
</tbody>
</table>

Arthur Samuel, a learner Checkers player, is one of the earliest and most important researchers on Checkers learning programs. From 1947 to 1967, his programs were based on many different methods of machine learning.

- Linear evaluation learning through self-play
- Nonlinear evaluation learning through book moves.
In linear evaluation learning, Samuel tuned the coefficients by positioning two copies of the Checkers programs named Alpha and Beta to play against each other. At the opening, Alpha and Beta are identical. The only difference is that Beta keeps its weights fixed while Alpha continuously tunes its weights during the experiment. If Alpha beats Beta, Beta adopts Alpha’s evaluation on the next round of experiments. Otherwise, Alpha tries other conduct to tune its weight. Sometimes manual intervention is necessary if the learning course gets stuck.

Samuel familiarizes a new procedure to build nonlinear evaluation functions through book moves through signature tables that are multi-dimensional tables where each dimension is indexed by the value of some feature. Samuel used book moves to train these cells by collecting a “book” of board positions and the equivalent moves played by human masters.

**Chinook**

Chinook is the world’s sturdiest computer checkers player, and the second strongest checkers player in overall. As it is a highly optimised and specialized program. Chinook responds virtually instantly, and without its opening or endgame database.

In the case of Chinook, “knowledge” is generally encoded in terms of its evaluation function. Additionally, Chinook was able to correctly forecast the opponent's move 80% of the time. Chinook divides the game into five phases - opening, middle game, early endgame, late endgame and database.

A position evaluation is the linear sum of the 22 heuristics multiplied by their user-defined weights. As last phase has flawless knowledge it had no parameters but a database of 800 classic Grandmaster games to develop and automatically tune the program's knowledge. At the time of writing, the Chinook team had announced that they had succeeded in building the entire 10-piece database. Computing such huge databases
and efficiently representing them in a dense form that can be used in real-time are both challenging issues.

**Blondie24 and C0**

Al-Khateeb B. and Kendall G. devised is an approach to evolve self-learning game of Checkers player called C0 which was based on Fogel’s Blondie24. It is based on artificial neural networks to evolve game playing strategies for the game of checkers by introducing a league structure into the learning phase of a system based on Blondie24 to eliminate some of the randomness in the evolution. The best player obtained is tested against an evolutionary checkers program based on Blondie24.

This program introduced learning mechanism into the learning phase of the evolutionary checkers system. The best player obtained is tested against an implementation of an evolutionary checkers program and also against a player, which utilizes a round robin tournament. The evolutionary collected results are shown below in figure 4.52.

![Blondie 24 Evolu Checkers Results](image)

**Fig. 4. 52 Checkers Implementation Resulted– Blondie 24 based C0**

Figure no 4.53 shows functional weight values change over a range of 25 generations. The weight changes are smoothly incremental for minimum values and increasing for maximum fitness values.
When above two figures are compared for implementation of the board game of Checkers it is very evident that Blondie24 based evolutionary results show incremental growth but they are cumulative score of all pool members of the checkers collected set of games. While fig 4.53 Shows the fitness collected value of every generation and these individual generation set highlights evolutionary growth of Checkers positions which are based on genetic string based program implementation.

Thus it can be very well derived that minimum and maximum fitness shown by blue and red bars has better evolutionary learning generations after generations. This proves the efficacy of genetic string based feature centric evolutionary computation based machine learning.