# Checkers using a Co-evolutionary On-Line Evolutionary Algorithm

**Evan J. Hughes**
Dept. Aerospace, Power & Sensors,
Cranfield University, RMCS, Shrivenham,
Swindon, UK. SN6 8LA
ejhughes@iee.org

**Abstract- The game of checkers has been well studied and many computer players exist. The vast majority of these 'software opponents' use a minimax strategy combined with an evaluation function to expand game tree for a number of moves ahead and estimate the quality of the pending moves.**

**In this paper, an alternative approach is described where an on-line evolutionary algorithm is used to co-evolve move sets for both players in the game, playing the entire length of the game tree for each evaluation, thus avoiding the need for the minimax strategy or an evaluation function.**

**The on-line evolutionary algorithm operates in essence as a 'directed' Monte-Carlo search process and although demonstrated on the game of checkers, could potentially be used to play games with a larger branching factor such as go.**

## 1 Introduction

Traditionally, most computer game-playing engines have been based on the minimax strategy [1]. According to Shannon [1] if we can enumerate the number of routes open to the player to score a win, minus the number of routes that would lead the opponent to win for each of the possible moves from the current board state, we could select the optimal move to make at each stage of the game. Unfortunately there are far too many possible routes to win/loss/draw to allow the expectation of the outcome to be evaluated exactly.

The minimax strategy expands the game tree for a number of ply (10 will give strong play in checkers, but the deeper the better), and then evaluates the leaf nodes using an evaluation function as an approximation of the expectation that the board state can lead to a win. Minimax always assumes that the opponent will make the worst possible move against the computer player, therefore the path through the game tree that leads to the best guaranteed payoff is taken. The strategy is very cautious, with no risk-taking in play. Other effects such as the limited search horizon can prolong games by the algorithm not seeing promising moves that occur after a prolonged period of play. Additionally if the minimax search indicates that sacrificing a piece may not be detrimental to a win in the future, the algorithm is quite likely to sacrifice a piece immediately, when another move may look more promising to a human player. The overall result is that the minimax algorithm leads to game-play that is quite mechanical in nature and not necessarily very pleasing to a human opponent. Yet the minimax algorithm is extremely effective even with limited processing resources.

As the evaluation function can only ever be a sub-optimal approximation of the true expected value of a board state, much effort has been employed to make the minimax search more efficient in order to descend deeper into the game tree. The design of the evaluation function however remains critical to the performance of the player. Many traditional hand-tuned evaluation functions exist, based on expert knowledge (for example [1, 2]). Very successful evaluation functions based on artificial neural networks have also been developed through the use of evolutionary algorithms [3]. These evolutionary approaches evolve the evaluation function off-line using a process of co-evolution by playing neural networks from within the same evolutionary algorithm population against each other. The best neural network that results after a period of evolution is then built into the game-playing structure as the evaluation function.

A complementary approach to deciding 'which move should I make next' is based on a direct Monte-Carlo evaluation of the expectation of a win for each of the available moves. Minimax and evaluation functions do not feature as the Monte-Carlo approach makes sequences of random moves until the leaves of the game tree are reached. Over many Monte-Carlo trials based on random moves, the number of wins minus the number of draws is calculated and the probability of each move leading to a win can be assessed (Abramson's expected outcome [4]). The Monte-Carlo methods are being used very successfully in go [5], but often tens of thousands of Monte-Carlo trials are used at each board location making the process slow.

In this paper an alternative approach is investigated where an on-line evolutionary algorithm (OLEA) is used to co-evolve a direct path through the game tree. The key difference between the Monte-Carlo methods and the on-line evolutionary algorithm is that the games are not played entirely at random, but are encoded in the chromosomes of the OLEA and evolved.

Two populations are used, one to represent the player who wishes to know the next move to play, and the other represents the opponent player. Each chromosome does not represent a means of assessing the current board state as in conventional game-playing algorithms (such as via an artificial neural network), but a single path from the current move, right through to a terminal leaf on the game tree giving a true win/draw/lose decision. The maximum number of genes is set by a draw being declared after 100 moves for each player, thus initially each chromosome has 100 genes, one to represent the move to make at each turn.

Co-evolution is used between the two populations to try to evolve an effective play sequence. Although the sampling at the leaves of the game-tree is exceptionally sparse, the hypothesis is that the genes which are soon to be used should still represent good moves that lead to reasonably promising regions of game-play. The second population is needed to provide an opponent to allow move sequences to be evaluated. The concept is to exploit co-evolution to evolve 'tough' sequences to play against, hopefully generating sequences that are stronger than the true opponent, leading to evolution of a playing sequence that will score a win.

As the entire game-tree is examined, it is hoped that the 'horizon effects' seen with minimax play will not be encountered and also that the process should be usable on games such as go with a very large branching factor.

Section 2 details the structure of the evolutionary algorithm. Section 3 describes the experimental process to examine the behaviour of the new algorithm against a traditional minimax player. Section 4 presents the experimental results and section 5 concludes.

## 2 On-Line Co-evolutionary Algorithm

### 2.1 Algorithm Structure

Figure 1 shows a representation of the co-evolutionary on-line evolutionary algorithm structure.

The basic concept is to run the evolutionary algorithm (EA) for a small number of generations before extracting the best-performing chromosome, allowing a decision on the next move to be made. The final population is used as the new population for the next move to be generated, etc. Thus a rolling population is used, evolving throughout the game. Similar co-evolutionary on-line evolutionary algorithms have been used successfully for other dynamic problems [6]. With the current chromosome representation, gene 1 corresponds to the next move to make. This is output and the board state updated. The EA then waits until the opponent makes its move, and the board state is updated accordingly. All chromosomes in both players are then cropped to remove the first (now played) gene, and the remaining chromosome segments are then used for a small number of generations (with selection and mutation) to generate the next move to play.

### 2.2 Chromosome Structure

The genetic representation is initialised as a sequence of 100 genes, one per move. Each gene is a real value in the range [0,1). The gene is decoded into a move by multiplying the real value by the total number of moves currently available, rounding to the nearest integer and using this as an index for a single move from the list of available moves. Although this prototype representation is highly epistatic, with future moves dependent on the current move choice (the number of available moves at each node in the game-tree will vary significantly), it serves to examine whether the co-evolutionary approach can produce useful game-play.

After each move is made, the first gene is cropped from all chromosomes, resulting in a chromosome structure that shortens as the game progresses.

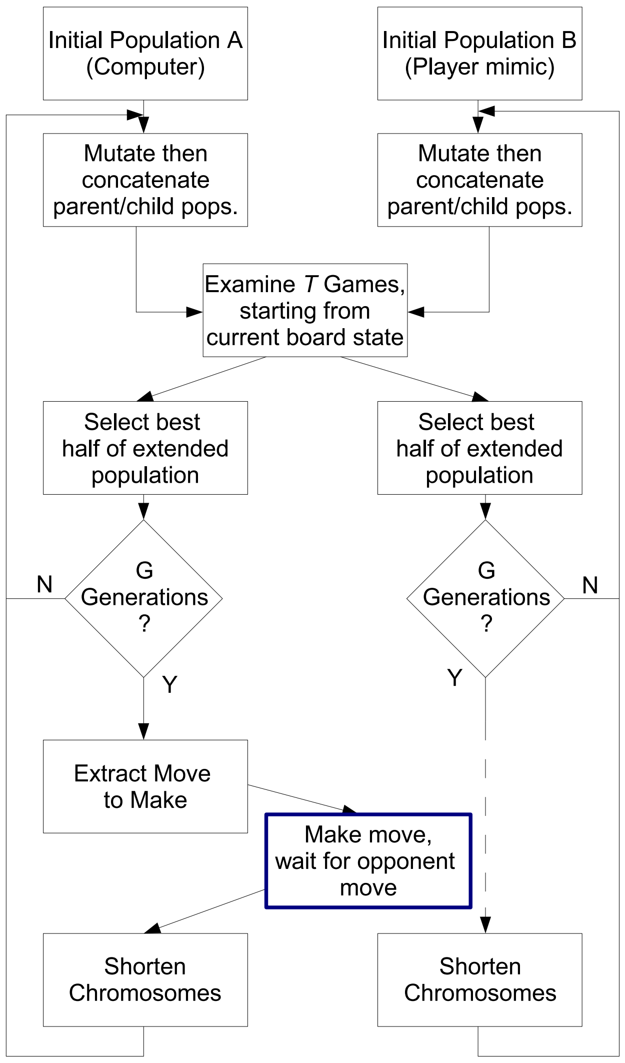
Initial Population A (Computer)
Initial Population B (Player mimic)
Mutate then concatenate parent/child pops.
Mutate then concatenate parent/child pops.
Examine *T* Games, starting from current board state
Select best half of extended population
Select best half of extended population
N
G Generations ?
G Generations ?
N
Y
Y
Extract Move to Make
Make move, wait for opponent move
Shorten Chromosomes
Shorten Chromosomes

Figure 1: Structure of co-evolutionary on-line evolutionary algorithm playing engine.

### 2.3 Objectives

The objective for optimisation is to choose chromosomes that win in the shortest possible time, or lose in the most moves possible in an attempt to force a draw.

A draw is scored as zero. A win after 99 moves scores 1, after 98 moves scores 2 etc. A loss after 99 moves scores -1, after 98 moves scores -2, etc.

Each chromosome plays a small number of games, $T$, against sequences drawn at random from the opponent population, but arranged so every chromosome in both the player and opponent population plays exactly $T$ games in each generation. Each chromosome is given an average score based on playing games against the $T$ opponent sequences. The score is to be maximised by the optimisation process. Every chromosome is re-evaluated every generation (i.e., the objective scores for unmodified chromosomes are not copied between generations to reduce computational time as the stochastic nature of the co-evolution produces very noisy objective values).

### 2.4 Selection and Reproduction

For each player, from a population of $P$, $P$ new solutions are created using a simple mutation-only evolutionary programme, adding Gaussian distributed noise with zero mean and a standard deviation of $0.7(1-(g-1)/G)$ where $G$ is the total number of generations to be performed and $g$ is the current generation number $g \in [1, G]$ (both on a per-move basis). The genes are then cropped if necessary to lie within the bounds [0,1). The initial standard deviation of 0.7 has been chosen through experimentation to give reasonable results, but is unlikely to be optimum.

The parents and child populations are concatenated and the $2P$ chromosomes are played against the opponent population (which undergoes a similar process) to calculate the objective scores. The best $P$ solutions are then retained for the next generation.

## 3 Playability Experiments

To test how effective the on-line EA approach is as a game-playing engine, a number of experiments have been devised. The key aspect of the experiments is to examine the gross effects on the level of competence of play of each of the EA parameters of population size, number of generations per ply and the number of games averaged to calculate the objective score.

The on-line EA approach was played against a simple piece difference strategy, combined with a minimax search algorithm to both 2 and 4-ply. The MTD($f$) [7] variant of minimax was used but without iterative deepening. In the piece difference, the kings were weighted as being equal to 1.3 men, as had evolved in other evaluation schemes [8]. A small random number was added to the final evaluation score that lay in the range [-0.25,0.25] and equates to plus/minus a quarter of a man. The small amount of randomness helps to prevent the play getting stuck in cyclic moves (such as one piece left in a corner). If a game reached 100 moves made by each player, a draw was recorded.

Table 1: First experimental Configuration

| Ref. | Pop. | Gens. | $T$ | Tot. games |
|---|---|---|---|---|
| aa | 25 | 20 | 1 | 1000 |
| ab | 50 | 10 | 1 | 1000 |
| ac | 100 | 5 | 1 | 1000 |
| ad | 500 | 1 | 1 | 1000 |
| ae | 5 | 20 | 5 | 1000 |
| af | 10 | 10 | 5 | 1000 |
| ag | 20 | 5 | 5 | 1000 |
| ah | 100 | 1 | 5 | 1000 |
| ai | 10 | 5 | 10 | 1000 |
| aj | 10 | 1 | 50 | 1000 |
| ba | 10 | 20 | 5 | 2000 |
| ca | 20 | 20 | 10 | 8000 |
| cb | 200 | 20 | 1 | 8000 |

A total of 100 games each as black and white was played in each experiment as a check to identify any underlying bias in playability.

The experimental strategy was to play algorithms with different combinations of the three parameters under investigation, but maintaining the same number of games played per ply. Thus the tradeoffs between the different algorithm configurations can be examined.

Table 1 details the first set of experiments performed. The first column is a reference for each experiment and the second column is the size of the working population used to represent each player. The third column is the number of generations performed at each move and the fourth column is the number of opponents played to calculate an aggregate objective value. The final column details the total number of game evaluations per player move.

During the evaluation of the first set of experiments, it was noticed that gene 1 in the chromosome that had the best score did not always represent the most common move in gene 1 of the entire population. It was decided to perform a second set of experiments where the move to make was the move that was most commonly selected from the first gene of all the chromosomes in the population. If multiple moves have the same frequency of occurrence in the population, the tie is broken by using the mean scores for the chromosomes that will produce the specific moves. This alternative move selection method has more in common with a conventional Monte-Carlo system.

Table 2 details the second set of experiments performed. The first column is a reference for each experiment and the second column is the size of the working population used to represent each player. The third column is the number of generations performed at each move and the fourth column is the number of opponents played to calculate an aggregate objective value. The fifth column is the number of ply the minimax opponent used. The final column details the total number of game evaluations per player move.

The second set of experiments concentrated on playing at higher ply, and also experiment *dd1* is against a player which makes moves at random to give a baseline indication

Table 2: Second experimental Configuration

| Ref. | Pop. | Gens. | $T$ | ply | Tot. games |
|---|---|---|---|---|---|
| da | 5 | 100 | 1 | 4 | 1000 |
| db | 10 | 50 | 1 | 4 | 1000 |
| dc | 20 | 25 | 1 | 4 | 1000 |
| dd1 | 25 | 20 | 1 | rand | 1000 |
| dd2 | 25 | 20 | 1 | 4 | 1000 |
| dd3 | 25 | 20 | 1 | 6 | 1000 |
| de | 50 | 10 | 1 | 4 | 1000 |
| df | 100 | 5 | 1 | 4 | 1000 |
| dg | 500 | 1 | 1 | 4 | 1000 |
| dh | 5 | 20 | 5 | 4 | 1000 |
| di | 10 | 10 | 5 | 4 | 1000 |
| dj | 20 | 5 | 5 | 4 | 1000 |
| dk | 100 | 1 | 5 | 4 | 1000 |
| dl | 10 | 5 | 10 | 4 | 1000 |
| dm | 10 | 1 | 50 | 4 | 1000 |
| ea | 10 | 20 | 5 | 4 | 2000 |
| fa | 20 | 20 | 10 | 4 | 8000 |
| fb1 | 80 | 50 | 1 | 6 | 8000 |
| fb2 | 80 | 50 | 1 | 8 | 8000 |
| fb3 | 80 | 50 | 1 | 10 | 8000 |
| fc | 100 | 40 | 1 | 6 | 8000 |
| fd1 | 200 | 20 | 1 | 4 | 8000 |
| fd2 | 200 | 20 | 1 | 6 | 8000 |

of performance against a random player. Experiment $fb2$ is the results of 50 games for each colour and at 8-ply the minimax algorithm has a similar processing time averaging at approximately 2.2 seconds per move. Experiment $fb3$ is the result of 25 games as both black/white and at 10-ply, the minimax algorithm is taking significantly longer on average to make a move (average approximately over 5 seconds per move, but with variations from 0.05 seconds to 22.4 seconds observed).

## 4 Results

### 4.1 Performance

The results of the experiments for Win:Draw:Loss (referenced to the EA approach) are as shown in table 3 for both 2-ply and 4-ply with the experimental configuration detailed in table 1. The results in table 4 correspond to the experimental configuration shown in table 2 and the OLEA uses the modified move selection process.

Most of the results are from only 100 trials for each colour and as such are subject to significant variation, however it is possible to infer general trends within the results that will help to tune the playing performance in the future.

### 4.2 First experiment set - Basic move selection

Table 3 shows the results of the first set of experiments. Comparing trials $aa, ab, ac, ad$, it is noticeable that the game-playing performance generally gets worse as the number of generations at each ply reduces, with the best performance where the population size is approximately the same

Table 3: Experimental Results for basic OLEA player

| | | 2-Ply | | | 4-Ply | | |
|---|---|---|---|---|---|---|---|
| | EA play | W | D | L | W | D | L |
| aa | B | 29 | 34 | 37 | 5 | 15 | 80 |
| | W | 36 | 19 | 45 | 4 | 9 | 87 |
| ab | B | 22 | 27 | 51 | 1 | 7 | 92 |
| | W | 19 | 19 | 62 | 5 | 6 | 89 |
| ac | B | 15 | 19 | 66 | 1 | 3 | 96 |
| | W | 19 | 11 | 70 | 1 | 0 | 99 |
| ad | B | 9 | 5 | 86 | 0 | 1 | 99 |
| | W | 11 | 4 | 85 | 1 | 0 | 99 |
| ae | B | 31 | 35 | 34 | 2 | 9 | 89 |
| | W | 28 | 31 | 41 | 2 | 7 | 91 |
| af | B | 33 | 38 | 29 | 2 | 19 | 79 |
| | W | 29 | 35 | 36 | 4 | 15 | 81 |
| ag | B | 28 | 31 | 41 | 2 | 10 | 88 |
| | W | 40 | 21 | 39 | 10 | 6 | 84 |
| ah | B | 23 | 17 | 60 | 4 | 1 | 95 |
| | W | 27 | 11 | 62 | 1 | 1 | 98 |
| ai | B | 32 | 35 | 33 | 3 | 12 | 85 |
| | W | 26 | 21 | 53 | 8 | 9 | 83 |
| aj | B | 26 | 29 | 45 | 4 | 6 | 90 |
| | W | 27 | 18 | 55 | 2 | 4 | 94 |
| ba | B | 40 | 36 | 24 | 5 | 19 | 76 |
| | W | 42 | 39 | 19 | 9 | 20 | 71 |
| ca | B | 51 | 34 | 15 | 9 | 23 | 68 |
| | W | 43 | 37 | 20 | 15 | 30 | 55 |
| cb | B | 25 | 23 | 52 | 5 | 7 | 88 |
| | W | 21 | 32 | 47 | 3 | 12 | 85 |

as the number of generations (experiment $aa$). Experiments $ca$ and $cb$ also show the same trend.

Comparing $aa$ and $ai$, it is noticeable that the number of games to form the aggregated objective value has little effect on the playing performance of the algorithm. Configuration $af$ appears to be the most effective, suggesting that aggregating a small number of games may be beneficial using the ‘select the best performer’ strategy for choosing the next move to play on the board.

Trial $ca$ showed best overall performance and is balanced in population size and generations (20 each) and has a reasonably small number of evaluations. But with 8 times as many games played as experiment $af$, shows playing performance improves with increased processing.

Trial $af$ shows performance that can be considered equivalent to the 2-ply minimax player with approximately the same number of wins and losses. Against the 4-ply player, configuration $af$ struggles. Player configuration $ca$ can be considered better than the 2-ply player, but not as good as 4-ply. Processing wise, the evolutionary approach is not quick, and is far slower than a 4-ply piece counting minimax player, although the code is not optimised and could be accelerated considerably.

### 4.3 Second experiment set - Modified move selection

Table 4 shows the results of the second set of experiments where the modified move selection process was applied. Comparing trials $dd2, de, df, dg$ and $dh, di, dj, dk$, with both one and five games averaged, it is noticeable that the game-playing performance generally gets worse as the number of generations at each ply reduces, with the best performance where the population size is approximately the same as the number of generations (experiment $dd2$). Experiments $fb1$ with respect to $fc$, and $fc$ with respect to $fd2$ also show the same trend.

Experiments $di$ and $dj$ are interesting as from the previous paragraph, $di$ would have been expected to be better. It is believed that with the low population size of only 10 in $di$, it cannot represent all the possible move combinations in some board configurations (especially when kings are in play) and so is hampered. This feature of the algorithm was also observed in $ae, af, ag$.

Comparing $df$ and $dk$, it is noticeable that the number of games to form the aggregated objective value (given a fixed population size) has a large impact on the playing performance of the algorithm. Configuration $dd2$ appears to be the most effective (with 1000 games per move calculation), suggesting that playing only a single game but using more generations may be beneficial using the ‘select the most common move’ strategy for choosing the next move to play on the board.

Trial $fb2$ showed best overall performance on the 4-ply scores and has a number of generations which is approximately two thirds of the population size (population 80, 50 generations), and only uses a single game trial to calculate each objective value. Experiments $fb1, fb2, fb3, fb4, fb5$ show that the game playing ability is strong, even against 10-ply minimax players.

Table 4: Experimental Results for modified OLEA player

| | EA play | ply | W | D | L |
|---|---|---|---|---|---|
| da | B | 4 | 4 | 16 | 80 |
| | W | | 3 | 21 | 76 |
| db | B | 4 | 3 | 22 | 75 |
| | W | | 8 | 25 | 67 |
| dc | B | 4 | 10 | 17 | 73 |
| | W | | 12 | 18 | 70 |
| dd1 | B | Rand | 100 | 0 | 0 |
| | W | | 100 | 0 | 0 |
| dd2 | B | 4 | 12 | 34 | 54 |
| | W | | 14 | 18 | 68 |
| dd3 | B | 6 | 1 | 2 | 97 |
| | W | | 4 | 2 | 94 |
| de | B | 4 | 10 | 22 | 68 |
| | W | | 8 | 24 | 68 |
| df | B | 4 | 0 | 25 | 75 |
| | W | | 5 | 18 | 77 |
| dg | B | 4 | 1 | 6 | 93 |
| | W | | 0 | 6 | 94 |
| dh | B | 4 | 2 | 16 | 82 |
| | W | | 3 | 18 | 79 |
| di | B | 4 | 7 | 21 | 72 |
| | W | | 9 | 17 | 74 |
| dj | B | 4 | 8 | 24 | 68 |
| | W | | 7 | 25 | 68 |
| dk | B | 4 | 0 | 3 | 97 |
| | W | | 0 | 4 | 96 |
| dl | B | 4 | 2 | 27 | 71 |
| | W | | 4 | 13 | 83 |
| dm | B | 4 | 1 | 1 | 98 |
| | W | | 0 | 2 | 98 |
| ea | B | 4 | 7 | 26 | 67 |
| | W | | 11 | 23 | 66 |
| fa | B | 4 | 16 | 37 | 47 |
| | W | | 27 | 42 | 31 |
| fb1 | B | 2 | 75 | 19 | 6 |
| | W | | 77 | 19 | 4 |
| fb2 | B | 4 | 29 | 35 | 36 |
| | W | | 30 | 29 | 41 |
| fb3 | B | 6 | 8 | 16 | 76 |
| | W | | 11 | 22 | 67 |
| fb4 | B | 8 | 1 | 7 | 17 |
| | W | | 0 | 13 | 12 |
| fb5 | B | 10 | 0 | 24 | 26 |
| | W | | 0 | 18 | 22 |
| fc | B | 6 | 6 | 17 | 77 |
| | W | | 10 | 18 | 72 |
| fd1 | B | 4 | 12 | 39 | 49 |
| | W | | 33 | 27 | 40 |
| fd2 | B | 6 | 0 | 13 | 87 |
| | W | | 5 | 14 | 81 |

Trial $fb2$ shows performance that can be considered equivalent to the 4-ply minimax player with approximately the same number of wins and losses. Against the 6-ply player ($fb3$), the configuration is not as strong, but still scores wins. Against the 8-ply player ($fb4$), again the OLEA has scored a win (results are from 25 games each colour only), but the processing time of the OLEA and the minimax algorithm are equivalent. Against the 10-ply player ($fb5$, results are from 25 games each colour only) the OLEA is drawing and losing in equal proportions, with no wins scored. However, the OLEA takes a consistent 2.2 seconds per move, whereas at 10-ply, some moves were taking over 20 seconds to play. A small number of games were played at 12-ply, and the OLEA both drew and lost. Unfortunately, although the OLEA was taking just over 2 seconds per move, some of the minimax moves were taking more than 5 minutes each and a reasonable number of games have yet to be gathered.

At low ply, the OLEA play is not as powerful as minimax for a given processing load. However, the OLEA is still capable of playing higher ply players and scoring draws. It is hoped that in future work looking at games such as chess and go with higher branching factors, the OLEA may become more efficient when compared to the time taken for deep minimax searches.

Overall, there also appears to be a slight bias towards playing second as the white player. The exact reason for the bias is unclear, but it may be related to the uneven distribution of opening moves being played by the minimax player.

### 4.4 Example Game

The following shows an example output, played to a win, from experiment $fb4$ playing black against the 8-ply player. The moves are as referenced by the board structure shown in figure 2. The opening 4 moves appear to be a variant of the *Double Corner*.

```
1) 9-14 22-17
2) 11-15 25-22
3) 8-11 29-25
4) 4-8 23-18
5) 14-(18)-23 (forced capture)
27-(23)-18
6) 12-16 17-13
7) 8-12 26-23
8) 16-20 30-26
9) 20-(24)-27 (forced capture)
31-(27)-24 (forced capture)
10) 11-16 18-(15)-11 (forced capture)
11) 16-20 23-18
12) 20-(24)-27 32-(27)-23 (forced
capture)
13) 7-(11)-16 (forced capture) 28-24
14) 16-19 24-(19)-15
15) 10-(15)-19 (forced capture)
23-(19)-16 (forced capture)
16) 12-(16)-19 (forced capture) 22-17
17) 19-24 26-23
18) 24-27 23-19
```

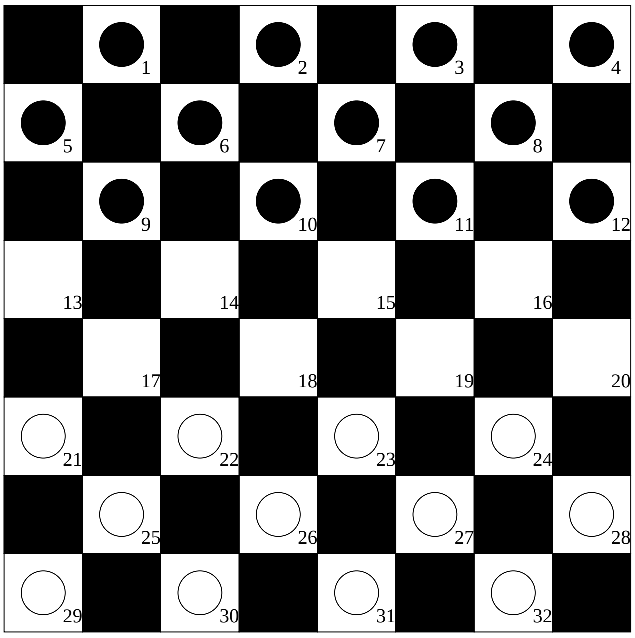

Figure 2: Board Layout

```
19) 27-31 19-16
20) 31-26 25-22
21) 26-23 18-15
22) 23-19 15-11
23) 19-(16)-12 (forced capture) 11-8
24) 3-7 8-4
25) 7-10 22-18
26) 6-9 13-(9)-6 (forced capture)
27) 2-(6)-9 (forced capture) 17-13
28) 10-14 13-(9)-6 (forced capture)
29) 1-(6)-10 18-(14)-9 (forced capture)
30) 5-(9)-14 (forced capture) 4-8
31) 12-(8)-3 (forced capture) 21-17
(forced move)
32) 14-(17)-21 (forced capture)
```

### 4.5 First Move Analysis

The first move made by each player as black was also recorded. The percentage that each of the strategies plays has been analysed and a synopsis of the results are presented in Table 5, as referenced by the board structure shown in figure 2. The first column shows the seven opening moves, the second column shows the percentages each move was used for the strategy $af$, the third column shows the moves used by strategy $ca$. The fourth column shows the moves used by the minimax player at 2-ply and the final column for minimax at 4-ply.

The results are interesting with the EA player for strategy $af$ playing little bias in the opening move, suggesting that little information has been extracted from the game tree – only a population of 10 for 10 generations was used with 5 games aggregated for the objective.

However, strategy $ca$ is demonstrating more biased play, targeting 11-15, 11-16 and 12-16 more heavily. The 2-Ply

Table 5: Move percentages

| Move | EA(af) | EA(ca) | 2-Ply | 4-Ply |
|---|---|---|---|---|
| 11-15 | 9 | 15 | 9 | 6 |
| 9-14 | 16 | 8 | 13 | 10 |
| 11-16 | 13 | 22 | 17 | 23 |
| 10-15 | 14 | 6 | 11 | 7 |
| 10-14 | 14 | 13 | 25 | 27 |
| 12-16 | 19 | 23 | 12 | 4 |
| 9-13 | 15 | 13 | 13 | 23 |

player however did seem to favour 10-14 and 11-16 as opening moves. When extended to 4-ply, the bias became more extreme with 10-14, 11-16 and 9-13 being used heavily. The minimax player only plays a range of moves because of the quarter-man randomness that is added to the evaluation. Without the random element, the piece count will generate the same move every time.

## 5 Conclusions

This paper introduces a novel approach for computer game-playing based on the on-line evolutionary algorithm. The experiments have demonstrated a player equivalent to 4-ply performance (against a minimax algorithm using piece count), and that also exhibits play that is superior to the equivalent minimax ply player, such as an extended play horizon, as can be witnessed by the 4-ply equivalent player still managing to win a game against 8-ply minimax.

Although the player described in this paper is a rather crude proof-of-concept algorithm, future work is anticipated to include: speeding up the software to investigate higher levels of play and comparison to deeper minimax searches; better chromosome structures that have lower epistasis, possibly with a more ‘spatial’ representation of the moves, rather than a move index; techniques to incorporate the details of the opponents actual move, such as promoting the chromosomes that predicted the move that was actually made; seeding the initial populations with good opening book strategies; improved EA structures that are more tolerant of the noisy fitness function; and alternative objectives – weight win more heavily than loss, or take the minimum/maximum of the small number of games rather than averaging in an attempt to generate cautious/aggressive play.

## Bibliography

[1] Claude E. Shannon. Programming a computer for playing chess. *Philisophical Magazine (Series 7)*, 41(314):256–275, March 1950.

[2] J. Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer, Berlin, 1996.

[3] David B. Fogel. *Blondie24: Playing at the edge of AI*. Morgan Kaufmann, 2001.

[4] B. Abramson. Expected-outcome : a general model of static evaluation. In *IEEE transactions on PAMI*, volume 12, pages 182–193, 1990.

[5] B. Bouzy and B. Helmstetter. Monte carlo go developments. In Ernst A. Heinz, H. Jaap van den Herik, and Hiroyuki Iida, editors, *10th Advances in Computer Games*, pages 159–174, Graz, 2003. Kluwer Academic Publishers.

[6] E.J. Hughes and B.A.White. On-line evolutionary algorithm guidance for multiple missiles against multiple targets. In *16th IFAC Symposium on Automatic Control in Aerospace*, St. Petersburg, Russia, 14-18 June 2004.

[7] Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie de Bruin. Best-first fixed-depth minimax algorithms. In *Artificial Intelligence*, volume 87, pages 255–293, November 1996.

[8] Kumar Chellapilla and David B. Fogel. Anaconda defeats hoyle 6-0: A case study competing an evolved checkers program against commercially available software. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 857–863, IEEE Press, Piscataway, NJ, 2000.