Applying Co-Evolutionary Particle Swarm Optimization to the Egyptian Board Game Seega

Ashraf M. Abdelbar
Department of Computer Science
American University in Cairo
abdelbar@aucegypt.edu

Sherif Ragab
Department of Computer Science
American University in Cairo
s_ragab@aucegypt.edu

Sara Mitri
Department of Computer Science
American University in Cairo
smitri@aucegypt.edu

Abstract- Seega is an ancient Egyptian two-phase board game that, in certain aspects, is more difficult than chess. The two-player game is played on either a 5×5, 7×7, or 9×9 board. In the first and more difficult phase of the game, players take turns placing one disk each on the board until the board contains only one empty cell. In the second phase players take turns moving disks of their color; a disk that becomes surrounded by disks of the opposite color is captured and removed from the board. We have developed a Seega program that employs co-evolutionary particle swarm optimization in the generation of feature evaluation scores. Two separate swarms are used to evolve White players and Black players, respectively; each particle represents feature weights for use in the position evaluation. Experimental results are presented and the performance of the full game engine is discussed.

1 Introduction

Games such as chess [4], backgammon [15], checkers [13], Othello [1, 9], and Go [11, 2] have been of interest to the AI research community [16]. The ancient Egyptian board game of Seega is a challenging game that, in some ways, is more difficult than chess, and may even be comparable to Go in difficulty. Seega is a two-player game and is most frequently played on a 7×7 board, but can also be played on a 5×5 or a 9×9 board, with complexity increasing with board size. For a 5×5 board, which we use in this paper, the White and Black players each have 12 disks, colored white and black respectively.

The first game phase is considered the heart of the game and the one where the bulk of the skill is needed; the second game phase is considered easier and requires less skill than the first phase. In the first phase, players take turns placing one disk each anywhere on the board except in the central cell, until only the central cell remains empty. In the second phase, each player, in each turn, is allowed to move one disk a single step, vertically or horizontally, if it has an adjacent empty cell. A disk that becomes surrounded, vertically or horizontally, by disks of the opposite color is considered captured and is removed from the board. If a player has no moves, then he is forced to pass, but cannot elect to pass if he has at least one legal move. The game continues until one player loses because all, or all but one of his disks have been removed, or there is a draw because 40 moves have been made without any captures.

The game is difficult for a minimax-based lookahead strategy, especially for larger board sizes, because in the first phase, when the important decisions must be made, it is not feasible for the lookahead to reach into the second phase where the actual captures are made. The evaluation function therefore has to incorporate much more game knowledge than a chess or Othello evaluation function.

In this paper we use a minimax search that looks ahead a number of ply and then applies an evaluation function. The evaluation function uses 6 feature evaluators in the first phase and 9 in the second phase; a weighted linear combination is used to combine the features into a single position evaluation. The vector of coefficients for the feature combiner is evolved by co-evolutionary particle swarm optimization.

Two swarms of 64 particles each are used for the White and Black colors, respectively. In each iteration a number of tournaments are played between players of the White swarm and players of the Black swarm. The games to be played are distributed across a cluster of 24 Intel Zeon 2.2GHz processors. The performance of each particle is determined by its performance in playing against members of the other swarm.

In the following section we present a fuller description of Seega and its rules. In Section 3 we review PSO and co-evolutionary PSO. Section 4 describes the features that are extracted by the feature evaluators. Section 5 discusses implementation and results. A sample game is shown in Section 6. Additional discussion and future work directions are presented in Section 7.

2 Game of Seega

Seega is a two-player ancient Egyptian capture board game, developed in Roman Egypt, and is still being widely played in rural areas of Egypt.

The game is most often played on a checkered 7×7 board, with 24 white and 24 black pieces. An easier version of the game uses a 5×5 board, and a more difficult version uses a 9×9 board. In theory, the game can be played on an $r \times r$ board for any odd r.

The game consists of two phases. White starts the first phase, during which pieces are placed on the board by each player in turn until all the squares, but the central one, are occupied. Players may place pieces on any unoccupied square except the central one.

Black starts the second phase, the aim of which is to capture as many of the opponent's pieces as possible. The player that ends up with one or fewer pieces loses the game.

A player is allowed to move any of his pieces into a horizontally or vertically adjacent unoccupied square on a turn. Of course, for the first move of the second phase, there is no choice but to move a piece into the central square. A player that makes a capture is allowed to play again.

A player captures one of the opponent's pieces by enclosing it from two opposite sides (horizontal or vertical), but only when this is the result of a move. If a player has no legal moves available, the opponent may play again until a path is cleared for the other player.

Seega is difficult because during the first phase, the player needs to plan ahead to the second phase, even though looking ahead to the second phase is not feasible except at the very end of the first phase. The difficulty and skill of the game therefore lies in placing the pieces during the first phase in preparation for the second phase.

3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [7], first presented in [5, 6] by Kennedy and Eberhart, is an optimization technique inspired by the concept of swarms in nature, such as bird flocking, fish schooling or insect swarming. The idea is that "individual members of the school can profit from the discoveries and previous experience of all other members of the school during the search for food [6]."

In the algorithm each individual in the particle swarm (hereafter referred to as a *particle*) is represented as a n-dimensional vector $\vec{w}$ for which we seek some kind of optimum.

A neighborhood is defined on the population as some mapping from each particle to some subset of the population. Here, we use a hypercube-neighborhood topology; for a population of size 2^k, every particle is assigned a corner of a k-dimensional hypercube. Two particles are said to be neighbors if they are exactly one edge away from each other.

A velocity vector $\vec{v}$, which defines a particle's current motion through the weight-space, and a vector $\vec{b}$ containing the best solution vector seen so far, are kept track of for each particle. Furthermore, a score value for $\vec{v}$ and $\vec{b}$ are stored for each particle.

Preceding every iteration of the PSO algorithm is an evaluation phase, during which the scores of the current weight vector is determined. This is achieved by finding the best particle for every neighborhood and accumulating the particle's score.

The algorithm first checks whether a vector is better than the best seen so far. Next, the best neighbor in terms of score is found, whose vector is denoted by $\vec{p}$.

Then the vectors are updated according to the following equation:

$$\begin{aligned}\vec{v}_t &= m \cdot \vec{v}_{t-1} + \phi_1(\vec{p} - \vec{c}_{t-1}) + \phi_2(\vec{b} - \vec{c}_{t-1}) \\ \vec{c}_t &= \vec{c}_{t-1} + \vec{v}_t\end{aligned} \tag{1}$$

The momentum m can be used to control how 'light' particles are, i.e. how difficult it is to accelerate them. The parameters m, ϕ_1, and ϕ_2 define the kinetic behavior of particles.

3.1 Co-evolutionary PSO

The idea of having multiple parallel populations in an evolutionary algorithm was first introduced in [12], where it was applied to Genetic Algorithms (GA). Potter and De Jong suggested the use of a species to represent a sub-component of a particular solution and to evolve each such species as a regular GA. By amalgamating the components resulting from each sub-population, a final solution is created for the problem.

Similar models are proposed in [10, 14] for PSO. According to Shi and Krohling, each population is run using the standard PSO algorithm, using the other population as its environment.

We have adopted this method since it has been observed that in Seega one has an advantage by playing Black (starting the second phase of the game, see section 2). The strategies of a Black player would therefore tend towards exploiting that advantage and attacking, while a White player would be more inclined to perfecting its defense strategies.

Applied to the algorithm presented above, this means that two population weight vectors coexist, one for Black, the other for White. During the evaluation phase, tournaments take place between the two species among particles of the two populations. The rest of the algorithm remains unchanged, and is performed independently on each population.

4 Feature Evaluators

Our approach uses minimax search with a small lookahead. At the leaves of the minimax tree, board positions are assessed using several feature evaluator functions which quantitatively describe certain aspects of the board. These functions are either *atomic* or *configurations*. As defined in [3], atomic functions analyze one criterion.

Most atomic features are *bipolar*, meaning they return a real number between -1 and 1. The larger that number, the more characteristic its measuring is for Black and not for White and vice versa.

Take the feature *material* as an example. For a 5×5 board, each player has at most 12 pieces and at least 2 (otherwise the game is over). Subtracting the number of white pieces from the number of black ones gives a value between $12 - 2 = 10$ and $2 - 12 = -10$. We then divide this value by 10 to produce a value between ± 1.

Other features are *unipolar*, returning a value between 0 and 1. These features give a color-neutral evaluation of a certain board aspect. The "mass-distance" features are examples of this, which compute the centers of mass for both Black and White, and return the distance between them. The atomic features are:

- corners (f_1): Corner domination.
- borders (f_2): Border domination.
- clustering ($f_{3..6}$): This feature is implemented four times, for Black (f_3 and f_4) and White (f_5 and f_6), and for each of those in the horizontal (f_3 and f_5) and

the vertical (f_4 and f_6) direction. It simply counts the number of horizontally/vertically adjacent white/black pieces.

- massdist ($f_{7,8}$): This feature exists twice, for horizontal (f_7) mass-distance and for vertical (f_8) mass-distance: the center of mass is computed for each color, and the magnitude of the difference is returned.
- entrapment ($f_{9,10}$): This feature again exists for the horizontal (f_9) and vertical (f_{10}) orientations. It gives a measure of how a color dominates the outer part of the board, i.e. to what degree one color surrounds the other.
- material (f_{11}): Reflects the count of black vs. white pieces on the board.
- phase two start (f_{12}): This feature reflects how many captures Black will make on the first move in phase one – this feature turns out to be extremely important.
- black can start (f_{13}): This feature returns 0 if the four squares around the middle square are occupied by White (i.e. Black cannot make the first move), and returns 1 otherwise.

These are combined into higher level configurations through addition and/or multiplication.

The final score returned by the evaluation function is a linear combination of the results of those compound features. In vector notation this can be expressed as follows; For phase one,

$$s_1 = (\vec{w}_1)^T \vec{c}_1$$

and for phase two,

$$s_2 = (\vec{w}_2)^T \vec{c}_2$$

where $\vec{w}$ is the weight vector, $\vec{c}$ is the vector of compound features, and s is the board score. The vector $\vec{c}$ is computed from the basic features as follows:

$$\vec{c}_1 = \begin{pmatrix} f_{12} \cdot f_{13} \\ -\frac{1}{2}(f_9 \cdot f_7 + f_{10} \cdot f_8) \\ \frac{1}{2}(f_7 + f_8) \\ f_3 + f_4 - f_5 - f_6 \\ \frac{1}{2}(f_1 + f_2) \\ f_1 - f_2 \end{pmatrix}$$

and

$$\vec{c}_2 = \begin{pmatrix} f_{11} \\ -\frac{1}{2}f_{11}(f_3 + f_4) \\ -\frac{1}{2}f_{11}(f_5 + f_6) \\ -\frac{1}{2}(f_9 \cdot f_7 + f_{10} \cdot f_8) \\ (f_3 - f_4) \cdot (f_9 - f_{10}) \\ (f_5 - f_6) \cdot (f_9 - f_{10}) \\ \frac{1}{2}f_{11}(f_7 + f_8) \\ \frac{1}{2}(f_1 + f_2) \\ f_1 - f_2 \end{pmatrix}$$

These expressions were chosen to represent meaningful playing strategies. Take for example the expression $-\frac{1}{2}f_{11}(f_3 + f_4)$, which is the 2^{nd} entry of $\vec{c}_2$. Here, material (f_{11}) which is a bipolar feature is multiplied by the sum of vertical and horizontal clustering (unipolar). This means that for a negative value of f_{11} (which usually means White is winning) and a high clustering value for Black (a defense mechanism), the entire expression will return a positive number which in effect counts this as an advantage for Black.

The relative importance of these features is optimized by the weight vector $\vec{w}$. The values of these weights are determined by the PSO algorithm (see section 3) and are different for each player.

Often, several different board positions will have the same score. To prevent the minimax algorithm from always selecting the same one, a very small random bias is added to the score before it is returned by the evaluation function. In this way, a different game is played every time even if the weight vectors stay the same.

5 Implementation and Results

The program we developed in this paper can basically be divided into two sub-systems. The game engine that carries out a game, and the PSO-based evolutionary component. The evolutionary component uses the game engine in a master-slave relationship in order to evaluate the fitness of its particles.

The game engine implements a regular minimax search-tree algorithm to determine a best next move for a certain board position. It is based on an evaluation function, which assigns a score to a board position, based on which the minimax algorithm selects best moves from the leaf nodes of the search tree. The best scores are propagated upward to finally determine the best next move. See [8] for more details on the minimax algorithm.

To increase the efficiency of the program, the depth of the minimax tree was modeled as a linear function of the pieces on the board at any given point in time, ranging from 2 to 10-ply. For example, at the beginning of the first phase when it is not fruitful to look too far ahead, the tree goes only 2 levels deep. The depth increases proportionally to the number of pieces on the board. In the first phase, the depth of the search is determined using the following equation:

$$d = 2 + \lfloor p(b) \cdot 0.3 \rfloor$$

where d is the depth, and $p(b)$ represents the total number of pieces on a particular instance b of the board. In the second phase, the depths of the search tree range between 4 and 6, this time according to this equation:

$$d = 4 + \lfloor (p(b) - 12)^2 \cdot 0.02 \rfloor$$

This technique has increased the speed of the games tremendously and avoids any unnecessary calculation. Furthermore, it tries to exploit the game-tree search algorithm at the most critical point of the game, namely, towards the end of the first phase, where the search reaches a depth of 10-ply.

The evolutionary subsystem is meant to optimize the weights given to the evaluation function, to make it return meaningful scores. Two swarms of 2^6 particles are used, where each swarm is based on a 6-dimensional hypercube topology. Therefore, each particle has 6 neighbors, and the intersection of any two neighborhoods includes no more than one particle.

At the end of each iteration each particle plays 5 consecutive games (the use of a small random bias in the evaluation function, as described earlier, allows for the possibility of different outcomes) against its image in the other swarm and 5 consecutive games against each of the six neighbors of its other-swarm image. This is a total of $5 \times 64 \times 7 = 2240$ games that are played in each iteration.

Each game carried out by the game engine returns a score to its master. This score is calculated to incorporate not only who won or lost the game, but also other data, such as the number of pieces left on the board, as well as the number of moves made throughout the game. This is necessary, for example, to reward players who lost a well-fought game over a player who simply lost without any resistance. The score s is calculated as follows:

$$s = r(1.0 + \Delta p - \Delta m)$$
$$\Delta p = k \cdot e^{c_1 (\ln 1/2)(p-12)/12}$$
$$\Delta m = k \cdot e^{c_2 (\ln 1/2)(m-25)/25}$$

where r is the result of the game, either 1 if Black won, -1 if White won or 0 if the game resulted in a draw. Δp represents the margin added according to the number of pieces p that were left at the end of the game out of the possible 24 pieces (since we are using a 5×5 board) and Δm stands for the reward/punishment for the number of moves in a game. These were chosen to be exponential functions, opposed to linear functions to put an upper limit to their values. The other variables, k, c_1 and c_2 are constants adjusted accordingly.

The large number of games were executed on a 24-processor cluster of Intel Zeon 2.2 GHz processors, with 512 MB of RAM on each processor.

Figure 2 shows the development of relative fitness for Black over the course of 800 iterations. This reveals an increase in relative fitness over time for Black, increasing the gap separating Black and White even further.

The White population displayed a general tendency to play towards a draw. This also becomes apparent by looking at Figure 2, which visualizes the progress of Black's relative fitness over time (White's is the exact mirror image reflected along the y-axis). The first 200 iterations contain larger fluctuations, after which relative fitness becomes quite stable at a relative fitness value of about 425.

The relationship here is that White relatively quickly found a way to efficiently play towards a draw, making it more and more difficult for Black to still make a win. As a result, the population 'settled' on a local optimum, producing players that could go several iterations without losing a single time.

Seeing two trained weight vectors play against each other (see next section) gives further evidence of this. White tries to accumulate its pieces on one side of the board and to occupy corners and borders. In this way, it is encouraging a draw rather than being ambitious and hoping for a win.

Figure 1 also shows how Black's weights have evolved over the 800 epochs. Note that the initial values are random. For the first 500 epochs or so the weights are still being adjusted and fluctuate in the process, after which stagnation is reached, remaining stable for the remaining 300 epochs. This particular particle has decided that vertical clustering - both for Black and White - is extremely important (f_4 and f_6), whereas horizontal clustering (f_3 and f_5) are the least important features. Occupying borders (f_2) as well as corners (f_1) seems to be relatively important to this player.

The source code for the present version of the program will be made publicly available.

6 A Sample Game

This section shows a game as carried out on the game engine between two particles after 800 epochs, one from the White and one from the Black swarm, each with its own weight vector.

Below is a list of the moves made throughout the game: a white or a black circle signifies whose turn it is; one coordinate is given in the first phase of the game, which represents the position where the disk was placed; both source and destination coordinates are given for phase two.

The first board illustration (Figure 3) shows the board at the end of phase one of the game. The strategies the players are following are already quite obvious, especially White's. The white pieces are all clustered to one side of the board, showing the effect of both the clustering ($f_{3..6}$) - which can also be observed in Figure 1 as the most important feature - and the center of mass ($f_{7,8}$) features. It is also noticeable that Black has tried to ensure that it can start (f_{13}), but has not taken control of the first few captures of the game and two of its pieces are immediately captured at the beginning of phase two.

1: ○ *c1,*	2: ● *c5,*	3: ○ *a3,*	4: ● *e3,*
5: ○ *d1,*	6: ● *d5,*	7: ○ *e4,*	8: ● *a4,*
9: ○ *c2,*	10: ● *e5,*	11: ○ *e1,*	12: ● *a5,*
13: ○ *d2,*	14: ● *e2,*	15: ○ *b3,*	16: ● *c4,*
17: ○ *a1,*	18: ● *b4,*	19: ○ *b2,*	20: ● *a2,*
21: ○ *d4,*	22: ● *b5,*	23: ○ *b1,*	24: ● *d3,*

The next 26 moves are displayed below. Even though White has managed to capture 2 of Black's pieces right at the beginning of the second phase and could have easily driven the game to its advantage, White prefers to go into defensive mode and play for a draw by clustering its pieces and avoiding any intermingling with the black pieces. This is especially evident from the second board illustration in Figure 4, which does not differ much from the original position at the start of phase two.

25: ● *c4-c3,*	26: ○ *d4-c4,*	27: ○ *e4-d4,*	28: ○ *d2-d3,*
29: ● *e5-e4,*	30: ○ *d1-d2,*	31: ● *d5-e5,*	32: ○ *b3-c3,*
33: ● *c5-d5,*	34: ○ *a3-b3,*	35: ● *a4-a3,*	36: ○ *c1-d1,*
37: ● *d5-c5,*	38: ○ *c2-c1,*	39: ● *e5-d5,*	40: ○ *c1-c2,*

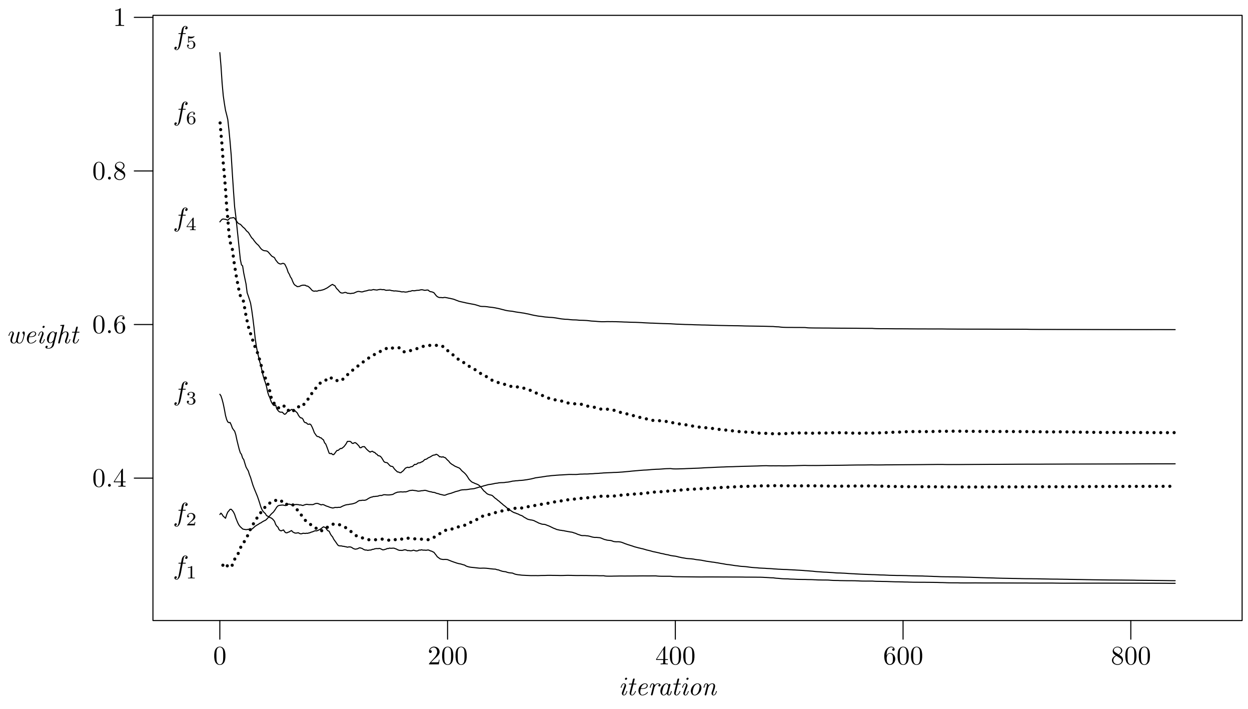

Figure 1: A black particle's feature weights for phase one

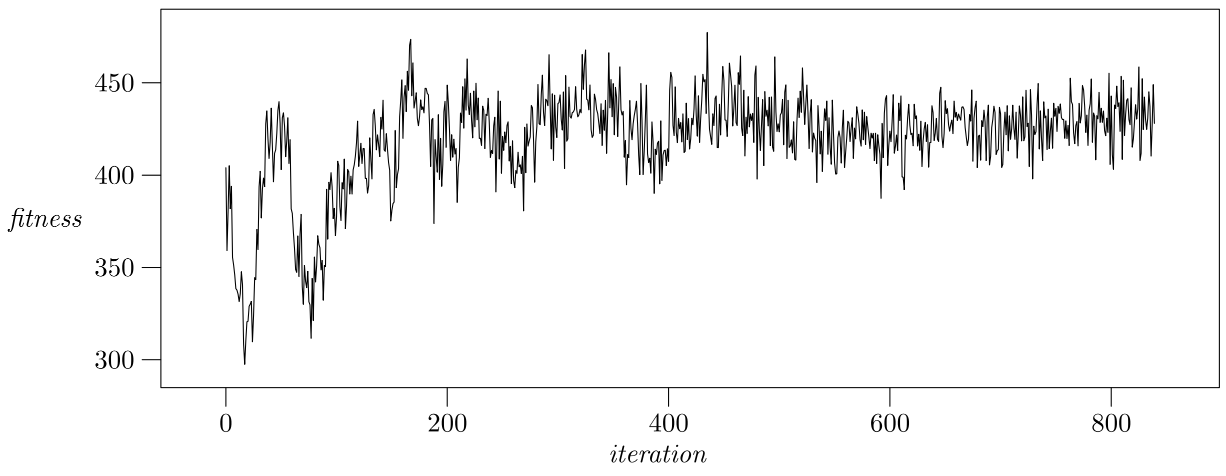

Figure 2: Black's relative fitness over time

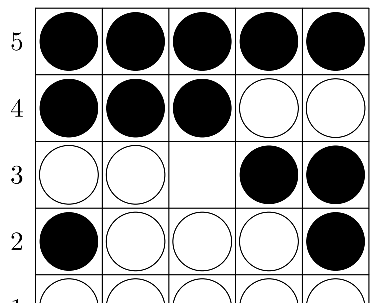

Figure 3: End of Phase One

41: ● *e4-e5,* 42: ○ *b1-c1,* 43: ● *b4-a4,* 44: ○ *b2-b1,* 45: ● *a4-b4,* 46: ○ *c2-b2,* 47: ● *b4-a4,* 48: ○ *c1-c2,* 49: ● *a4-b4,* 50: ○ *d1-c1,*

Insisting on its retreat, White waits till 40 capture-less moves have passed by simply moving back-and-forth, leading to the final result of a draw. The final board position is shown in Figure 5, again with only very slight changes that seem to be due to Black's efforts to attack.

This game is a clear example of the problems we are facing with the evolved White players, which we hope to tackle in future work.

51: ● *b4-a4,* 52: ○ *c1-d1,* 53: ● *a4-b4,* 54: ○ *d1-c1,* 55: ● *b4-a4,* 56: ○ *c1-d1,* 57: ● *a4-b4,* 58: ○ *b1-c1,* 59: ● *e5-e4,* 60: ○ *c1-b1,* 61: ● *a5-a4,* 62: ○ *d1-c1,*

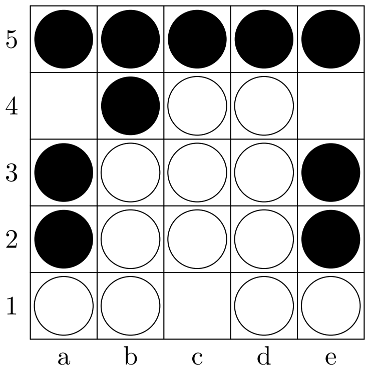

Figure 4: After 50 moves

63: ● *e4-e5,* 64: ○ *e1-d1,* 65: ● *e2-e1,* 66: ○ *d2-e2,* 67: ● *e3-e4* — draw!

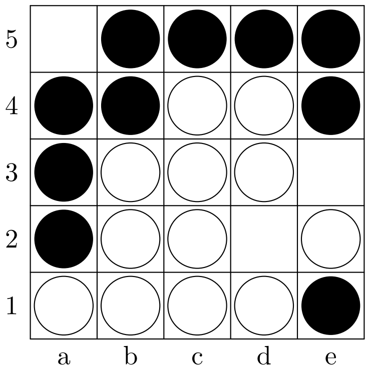

Figure 5: End of Game

7 Conclusions and Future Work Directions

This paper represents a first attempt at developing a PSO-based game engine for the ancient Egyptian board game Seega. The performance of the program against human players is found to improve as the number of PSO iterations increases. In the present work, we used the smaller 5×5 board, however, we would like to transfer to the 7×7 board once this size has been sufficiently mastered.

In the future, we would like to explore other tournament schemes with the aim of reducing the number of games used to evaluate the members of the co-evolutionary Black and White swarms, and also of increasing the amount of competition in the games played, by ensuring that players get more challenging partners.

We are having more difficulty evolving White players than Black. The game is of course asymmetric, with White playing first in the first phase but Black playing first in the second phase. As a result, one's strategy playing as White is very different from when playing as Black.

In the future, it would also be interesting to combine the PSO approach with a GA. The hybrid algorithm proposed in [10] integrates selection and crossover operators into the PSO algorithm. This is done at each time step: when the vectors of the particles are recalculated, certain particles from the swarm are selected and recombined with each other to create a new generation of particles. The idea behind this approach is to increase the variation between particles of each generation, which has the potential of improving performance for the Seega application.

Bibliography

[1] A.M. Abdelbar, and G. Tagliarini. Using neural network learning in an Othello evaluation function. *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 10, pp. 217–229, 1998.

[2] Burmeister, and Wiles, J. AI techniques used in computer Go. *Fourth Conference of the Australasian Cognitive Science Society*, Newcastle, 1997.

[3] Buro, M. From Simple Features to Sophisticated Evaluation Functions. *Proc. of the First Int. Conf. on Computers and Games (CG-98)*, Vol. 1558, pp. 126–145, 1998.

[4] Campbell, M., Hoane, A. J. Jr. and Hsu, F. Deep Blue. *Artificial Intelligence*, Vol. 134, pp.57–83, 2002.

[5] R.C. Eberhart, and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings International Symposium on Micro Machine and Human Science*, pp. 39-43, 1995.

[6] J. Kennedy, and R.C. Eberhart. Particle swarm optimization. In *Proceedings IEEE International Conference on Neural Networks*, Vol. IV, pp. 1942–1948, 1995.

[7] J. Kennedy, and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[8] Korf, R. E. Artificial Intelligence Search Algorithms. *Algorithms and Theory of Computation Handbook, CRC Press*, 1999.

[9] K.-F. Lee, and S. Mahajan. The development of a world-class Othello program. *Artificial Intelligence*, Vol. 43, pp. 21–36, 1990.

[10] Løvbjerg, M., Rasmussen, T. K. and Krink, T. Hybrid Particle Swarm Optimiser with Breeding and Subpopulations. *Proc. of the third Genetic and Evolutionary Computation Conference (GECCO-2001)*, Vol. 1, pp. 469–476, 2001.

[11] Müller, M. Computer Go. *Artificial Intelligence*, Vol. 134, pp. 145–179, 2002.

[12] Potter, M. A. and De Jong, K. A. A Cooperative Coevolutionary Approach to Function Optimization. *Proc. of the Third Conf. on Parallel Problem Solving from Nature*, pp 249–257, 1994.

[13] Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, Vol. 44, No. 1/2, pp. 207–226, 2000.

[14] Shi, Y. and Krohling, R. A. Co-evolutionary Particle Swarm Optimization to Solve Min-max Problems. *Honolulu, Hawaii USA. IEEE Congress on Evolutionary Computation*, 2002.

[15] Tesauro, G. TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, Vol. 6, pp. 215–219, 1994.

[16] Van Den Herik, H. J., Uiterwijk, J. W. H. M. and van Rijswijck, J. Games Solved: Now and in the Future. *Artificial Intelligence*, Vol. 134, pp. 277–311, 2002.