Evolutionary Algorithms for Board Game Players with Domain Knowledge

Kyung-Joong Kim and Sung-Bae Cho

Department of Computer Science, Yonsei University,
134 Shinchon-dong, Sudaemoon-Ku, Seoul 120-749, South Korea
{kjkim,sbcho}@cs.yonsei.ac.kr

Abstract. Incorporating a priori knowledge, such as expert knowledge, meta-heuristics, human preferences, and most importantly domain knowledge discovered during evolutionary search, into evolutionary algorithms has gained increasing interest in recent years. In this chapter, we present a method for systematically inserting expert knowledge into evolutionary board game framework at the opening, middle, and endgame stages. In the opening stage, openings defined by the experts are used. In this work, we use speciation techniques to search for diverse strategies that embody different styles of game play and combine them using voting for higher performance. This idea comes from the common knowledge that the combination of diverse well-playing strategies can defeat the best one because they can complement each other for higher performance. Finally, we use an endgame database. Experimental results on checkers and Othello games show that the proposed method is promising to evolve better strategies.

1 Introduction

There are two extreme approaches to develop game strategies. One is to use expert domain knowledge extensively in opening, middle and endgame stages. For example, Chinook, the best checkers player in the world, used huge opening and endgame databases with the expert's comments on feature selection and weight determination of evaluation function [1]. It shows very successful to get the world championship-level performance but they need much effort to do that. The other is to use an evolutionary computation to extract novel and good strategies without expert knowledge. The procedure of strategy induction is based on the competition among randomly generated individuals and genetic operations for guiding strategy search. For example, Fogel evolved master-level checkers players from the pure evolution [2].

Although both of them provide a way to create game strategies, they have a limitation to be a more practical one. To solve the problem, hybrid of both approaches was proposed and realized in many different games [3, 4, 5].

K.-J. Kim and S.-B. Cho: *Evolutionary Algorithms for Board Game Players with Domain Knowledge*, Studies in Computational Intelligence (SCI) **71**, 71–89 (2007)
www.springerlink.com
© Springer-Verlag Berlin Heidelberg 2007

The focus of the approach is to utilize expert knowledge in evolutionary algorithms. The domain knowledge can be exploited in many parts of evolutionary algorithms; genetic operators, selection, and competition. The incorporation of domain knowledge guides the search direction of evolutionary algorithms or reduces the search spaces and improves the performance of search. Although the incorporated knowledge is not complete compared to the knowledge-based approach, the evolutionary algorithm can be enhanced by the information.

Many board games including Othello, Go, Chess, and Checkers have similar properties. They are perfect information of games and there is no hidden information. Usually, each board game has three different stages: opening, middle, and endgame stages. In opening stage, each game has its own book moves. Expert players are very good at memorizing or organizing a huge number of opening moves. A small disadvantage in an early stage of game results in a great loss after the opening. In the middle stage, it is difficult to define the helpful knowledge explicitly. At the endgame stage, there are relatively small possible choices and it allows programs to calculate the win/draw/loss of the game perfectly. The endgame positions are memorized in a form of DB or perfectly solved in a few seconds.

Opening and endgame knowledge of board games are easily accessible through Internet though they are not enough to make world-level champion program. Board game associations of each country, world organization of board games and many basic introductory materials (books and internet website) provide well-defined knowledge about the games. Using the knowledge, an evolutionary induction approach for board games can be enhanced. Though evolutionary algorithm gives much freedom to the designer, it needs a long evolution time and sometimes, very simple tactics are not evolved or ignored. The idea of knowledge incorporation to the board game is summarized in the figure 1.

In this chapter, the representative two board games, Othello and Checkers, are used to illustrate the feasibility of the approach. Both of them are very well-known games that are frequently used in AI community. Opening and endgame knowledge are well-defined for the games. They are involved in the evolution of board evaluator represented neural networks and weights

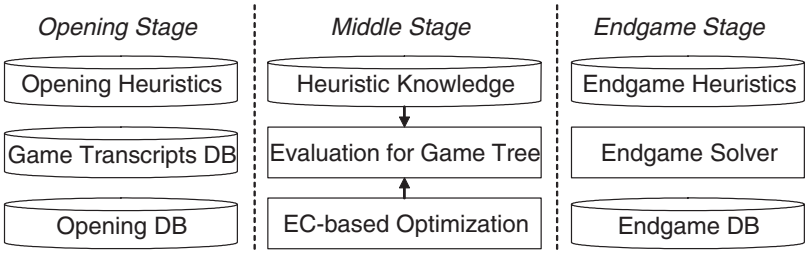


Fig. 1. Conceptual diagram of knowledge-incorporated evolutionary induction of strategies (EC = Evolutionary Computation)

matrix. In Checkers, neural network evaluator is evolved with opening and endgame knowledge. In Othello, weight piece counter is evolved with opening knowledge.

2 Related Works

2.1 Traditional Game Programming

Usually a game can be divided into three general phases: the opening, the middle game, and the endgame. Entering thousands of positions in published books into the program is a way of creating an opening book. The checkers program Colossus has a book of over 34,000 positions that were entered manually [6]. A problem with this approach is that the program will follow published play, which is usually familiar to the humans [7]. Without using an opening book, some programs find many interesting opening moves that stymie a human quickly. However, they can also produce fatal mistakes and enter a losing configuration quickly because a deeper search would have been necessary to avoid the mistake. Humans have an advantage over computers in the opening stage because it is difficult to quantify the relevance of the board configuration at an early stage. To be more competitive, an opening book can be very helpful but a huge opening book can make the program inflexible and without novelty.

The one of important parts of game programming is to design the evaluation function for the middle stage of the game. The evaluation function is often a linear combination of features based on human knowledge, such as the number of kings, the number of checkers, the piece differential between two players, and pattern-based features. Determining components and weighting them require expert knowledge and a long trial-and-error tuning. Attempts have been made to tune the weights of the evaluation function through automated processes, by using linear equations, neural networks, and evolutionary algorithms and can compete with hand-tuning [5, 8].

In chess, the final outcome of most games is usually decided before the endgame and the impact of a prepared endgame database is not particularly significant. In Othello, the results of the game can be calculated in real-time if the number of empty spaces is less than 26. In these two games, the necessity of the endgame database is very low, but in checkers the usage of an endgame database is extremely beneficial. Chinook has perfect information for all checkers positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. These databases are now available for download. The total download size is almost 2.7GB (compressed) [9]. Recently, the construction of a 10-piece database has been completed.

2.2 Evolving Game Players

Checkers is the board game for which evolutionary computation has been used to evolve strategies. Fogel *et al.* have explored the potential for a co-evolutionary process to learn how to play checkers without relying on the usual inclusion of human expertise in the form of features that are believed to be important to play well [10, 11, 12, 13]. After only a little more than 800 generations, the evolutionary process generated a neural network that can play checkers at the expert level as designated by the U.S. Chess Federation rating system. In a series of six games with a commercially available software product named "Hoyle's Classic Games," the neural network scored a perfect six wins [14]. A series of ten games against a "novice-level" version of Chinook, a high-level expert, resulted in 2 wins, 4 losses, and 4 draws [15].

Othello is a well-known and challenging game for human players. Chong *et al.* applied Fogel's checkers model to Othello and reported the emergence of mobility strategies [16]. Wu *et al.* used fuzzy membership functions to characterize different stages (opening game, mid-game, and end-play) in the game of Othello and the corresponding static evaluation function for each stage was evolved using a genetic algorithm [17]. Moriarty *et al.* designed an evolutionary neural network that output the quality of each possible move at the current board configuration [19]. Moriarty *et al.* evolved neural networks to constrain minimax search in the game of Othello [20, 21]. At each level, the network saw the updated board and the rank of each move and only a subset of these moves were explored.

The SANE (symbiotic, adaptive neuro-evolution) method was used to evolve neural networks to play the game of Go on small boards with no preprogrammed knowledge [22]. Stanley *et al.* evolved a roving eye neural network for Go to scale up by learning on incrementally larger boards, each time building on knowledge acquired on the prior board [23]. Because Go is very difficult to deal with, they used a small size board, such as 7×7, 8×8, or 9×9. Lubberts *et al.* applied competitive co-evolutionary techniques of competitive fitness sharing, shared sampling, and a hall of fame to the SANE neuro-evolution method [24]. Santiago *et al.* applied an enforced subpopulation variant of SANE to Go and an alternate network architecture featuring subnetworks specialized for certain board regions [25].

Barone *et al.* used evolutionary algorithms to learn to play games of imperfect information – in particular, the game of poker [26]. They identified several important principles of poker play and used them as the basis for a hypercube of evolving populations. Co-evolutionary learning was used in backgammon [27] and chess [5, 28]. Kendall *et al.* utilized three neural networks (one for splitting, one for doubling down, and one for standing/hitting) to evolve blackjack strategies [29]. Fogel reported the experimental results of evolving blackjack strategies that were performed about 17 years ago in order to provide some baseline for comparison and inspiration for future research [30]. Ono *et al.* utilized co-evolution of artificial neural networks on a game

Table 1. Summarization of related-works in evolutionary games

Information	Game	Reference	Knowledge Incorporation
Perfect Information	Checkers	2	
	Othello	16	
		17	O
		19	
		20	
	Go	22	
	Chess	28	O
		5	O
	Kalah	31	O
Imperfect Information	Backgammon	27	
	Poker	26	O
	Blackjack	29	O
		30	O

called Kalah and the technique closely followed the one used by Chellapilla and Fogel to evolve the successful checkers program Anaconda (also known as Blondie24) [31]. Table 1 summarizes the related works. Fogel's checkers framework was used in other games such as [5, 16, 29] and [31]. Fogel *et al.* applied the framework to the game of chess and reported that the evolved program performed above the master level 5. More information about evolutionary game players is available at [32].

2.3 Othello and Evolution

Because the strategy of Othello is very complex and hard to study even for human, researchers use the game as a platform of AI research. Miikkulainen *et al.* used evolutionary neural network as a static evaluator of the board [19]. Because they used marker-based encoding for neural network, the architecture and weights were coevolving. In their work, they did not use game tree but the evolution generated mobility strategy, one of the important human strategies for the game. The analysis by the world champion Othello player, David Shaman, was attached and the result showed the possibility of evolutionary approach for the game. In other works, they used evolutionary neural network to focus the search of game tree [20, 21]. In the work, game tree was used to find the good move with static evaluator. The purpose of the evolution was to find a neural network that decided whether deep search was needed. By reducing the moves for deep search, it was possible to expand the search depth more than previous one.

Chong *et al.* used the same framework used in successful evolutionary checkers to evolve Othello strategies [16, 33]. Their work showed that the same architecture could be successful in another game (Othello). They discussed

that the reasons of success were spatial preprocessing layer, self-adaptive mutation, and tournament-style selection (coevolving). They evolved only the weights of fixed-architecture neural network and the concept of spatial preprocessing (dividing the board into a number of sub-boards) were used.

Sun *et al.* proposed dynamic fitness function for the evolution of weights of linear static evaluator for Othello [18]. The fitness function was changed according to the performance of the previous stage. They also investigated the expansion of chromosome structures to meet new challenges from the environment in the game of Othello [17]. In other work, they used multiple coaches (they were used for fitness evaluation) selected from local computer Othello tournament before evolution [34]. Sun's work was focused on increasing the diversity and self-adaptation of evolutionary algorithm given linear evaluation function. The purpose of evolution was to adjust the weights of features (position, piece advantage, mobility and stability) by expert.

Alliot *et al.* proposed a method for improving the evolution of game strategies [35]. They evolved weights of linear evaluation function (it was similar to the weight piece counter). The sharing scheme and sophisticated method for fitness evaluation were used. Because Othello's rule was so simple to understand, it was frequently used for educational purpose to teach student about the evolutionary mechanism [36, 37]. Smith *et al.* proposed co-evolutionary approach for Othello [38]. The fitness of the member in the population was evaluated by the competition results with other members. There were some works about reinforcement learning or self-teaching neural networks for Othello [39, 40]. The discussion about the relationships between reinforcement learning and the evolutionary algorithm can be found at [41].

3 Evolutionary Game Players with Domain Knowledge

3.1 Evolutionary Checkers Framework with Speciation

Usually, an evaluation function is the linear sum of the values of relevant features selected by experts. Input of the evaluation function is the configuration of the board and the output of the function is a value of quality. Designing the function manually needs expertise in the game and tedious trial-and-error tuning. Some features of the board evaluation function can be modeled using machine learning techniques such as automata, neural networks, and Bayesian networks. There are some problems for learning the evaluation function such as determining the architecture of the model and transformation of the configuration into numerical form.

The feed-forward neural network, which has three hidden layers comprising 91 nodes, 40 nodes, and 10 nodes, respectively is used as an evaluator. Multiple neural networks are generated from evolutionary algorithm such as speciation and they are combined to improve the performance. The board configuration is an input to the neural network that evaluates the configuration and produces

0	1	1	1
0	0	1	1
0	1	1	1
1	0	$-K$	0
0	$-K$	0	0
0	0	0	0
0	0	0	-1
0	-1	-1	-1

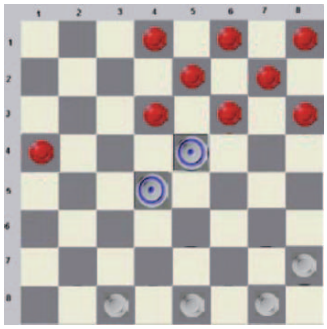


Fig. 2. An example of board representation. Minus sign means the opponent checkers and K means the King. The value of K is evolved with the neural networks

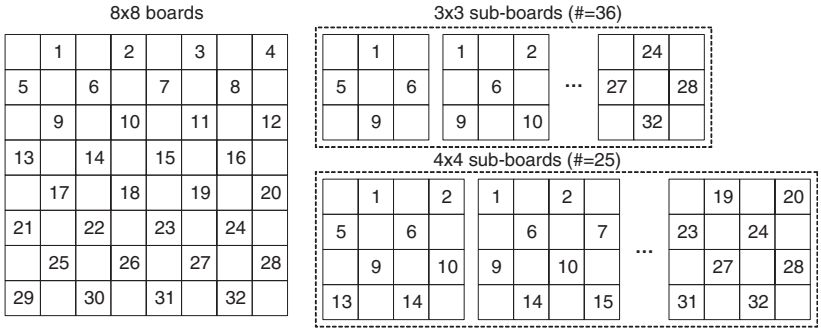


Fig. 3. An example of 4×4, and 3×3 sub-boards. In a checkerboard, there are 91 sub-boards (36 3×3 sub-boards, 25 4×4 sub-boards, 16 5×5 sub-boards, 9 6×6 sub-boards, 4 7×7 sub-boards, and 1 8×8 sub-board). This design gives spatial local information to the neural networks

a score representing the degree of relevance of the board configuration. For evaluation, the information of the board needs to be transformed into the numerical vectors. Each board is represented by a vector of length 32 and components in the vector could have a value of $\{-K, -1, 0, +1, +K\}$, where K is the value assigned for a king, 1 is the value for a regular checker, and 0 represents an empty square. Figure 2 depicts the representation of a board.

To reflect spatial features of the board configuration, sub-boards of the board are used as an input. One board can have 36 3×3 sub-boards, 25 4×4 sub-boards, 16 5×5 sub-boards, 9 6×6 sub-boards, 4 7×7 sub-boards, and 1 8×8 sub-board. 91 sub-boards are used as an input to the feed-forward neural network. Figure 3 shows an example of 3×3, 4×4, and 6×6 sub-boards. The sign of the value indicates whether or not the piece belongs to the player or the opponent. The closer the output of the network is to 1.0, the better the position is. Similarly, the closer the output is to -1.0, the worse the board. Figure 4 shows the architecture of the neural network.

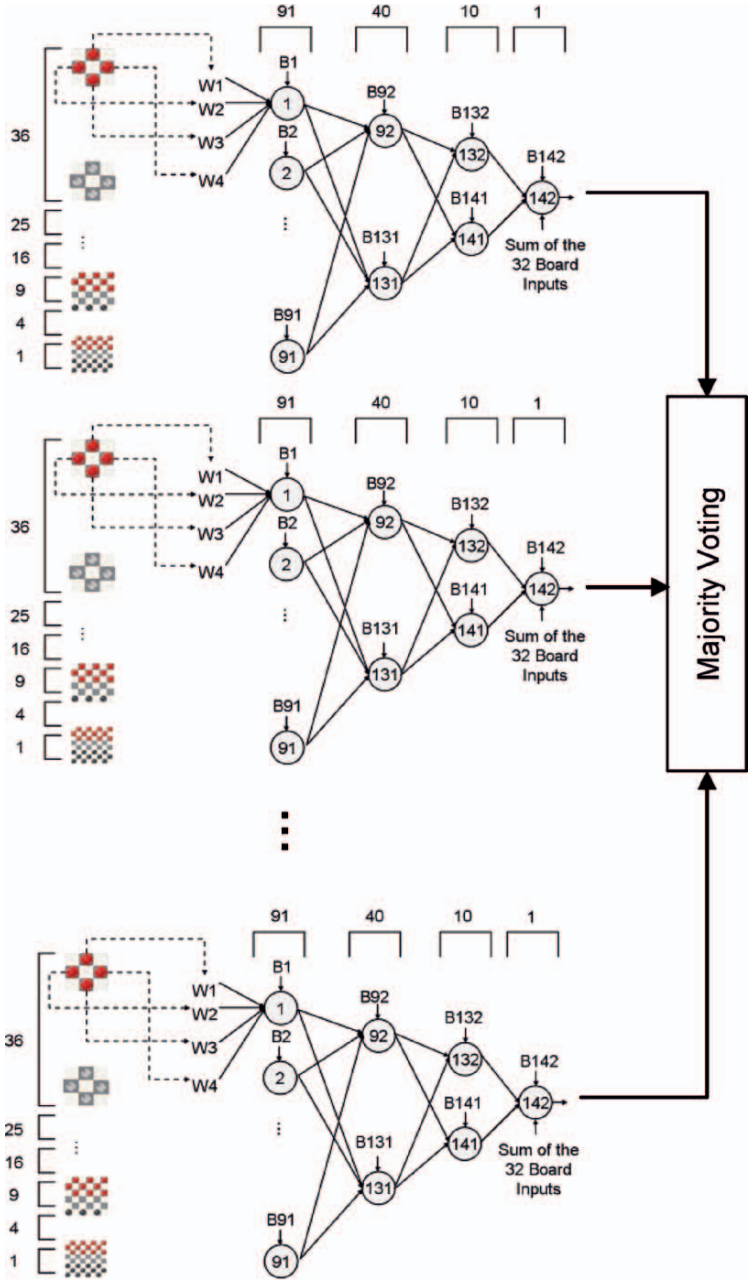


Fig. 4. The architecture of neural network. It is fully connected in the hidden-layers. One sub-board is transformed into the corresponding vector representation and used for the input of the neuron. The output of the neural network indicates the quality of the board configuration

The architecture of the network is fixed and only the weights can be adjusted by evolution. Each individual in the population represents a neural network (weights and biases) that is used to evaluate the quality of the board configuration. Additionally, each neural network has the value of K and self-adaptive parameters for weights and biases. An offspring $P'_i, i = 1, ..., p$ for each parent $P_i, i = 1, ..., p$ is created by

$$\sigma'_i(j) = \sigma_i(j) \exp(\tau N_j(0,1)),\ j = 1, ..., N_w$$

$$w'_i(j) = w_i(j) + \sigma'_i(j) N_j(0,1),\ j = 1, ..., N_w$$

where N_W is the number of weights and biases in the neural network (here this is 5046), $\tau = 1/\sqrt{2\sqrt{N_w}} = 0.0839$, and $N_j(0,1)$ is the standard Gaussian random variable resampled for every j. The offspring king value K' was obtained by

$$K'_i = K_i + \delta$$

where δ was chosen uniformly at random from $\{-0.1,\ 0,\ 0.1\}$. For convenience, the value K' was constrained in $[1.0,\ 3.0]$ by resetting to the limit exceeded when applicable.

In fitness evaluation, each individual chooses five opponents from a population pool and plays games with the players. Fitness increases by 1 for a win while the fitness of an opponent decreases by 2 for a loss. In a draw, the fitness values of both players remain the same. After all the games are played, the fitness values of all players are determined. Deterministic crowding algorithm is used to maintain the diversity of the population. After finishing evolution, the final population is clustered into species and the representative players from each species are combined to play.

3.2 Knowledge Incorporation into Evolutionary Checkers

As mentioned before, we can classify a single checkers game into three stages: opening, middle, and endgame stages. In the opening stage, about 80 previously summarized openings are used to determine the initial moves. In the middle stage, a game tree is used to search for an optimal move and an evolutionary neural network is used to evaluate leaf nodes of the tree. In the neural network community, it is widely accepted that the combination of multiple diverse neural networks can outperform the single network [42]. Because the fitness landscape of an evolutionary game evaluation function is highly dynamic, a speciation technique like fitness sharing is not appropriate. A crowding algorithm that can cope with a dynamic landscape is adopted to generate more diverse neural network evaluation functions. The performance of evolutionary neural networks for creating a checkers evaluation function has been demonstrated by many researchers 2. In the end game, an endgame database is used which indicates the result of the game (win/loss/draw) if the number of remaining pieces is smaller than a predefined number (usually from 2 to 10).

Opening Stage

The opening stage is the most important opportunity to defeat an expert player because trivial mistakes in the opening can lead to an early loss. The first move in checkers is played by red and there are seven choices (9-13, 9-14, 10-14, 10-15, 11-15, 11-16, and 12-16). These numbers refer the labels on the board (the top-left square is 1) and the X-Y means red moves a piece from position X to position Y. Usually, 11-15 is the best move for red but there are many other alternatives. They are described with specific names, such as Edinburgh, Double Corner, Denny, Kelso, Old Faithful, Bristol, and Dundee, respectively. For each choice, there are many well-established additional sequences which range in length from 2 to 10. The longest sequence is described as the White Doctor: 11-16, 22-18, 10-14, 25-22, 8-11, 24-20, 16-19, 23-16, 14-23, 26-19. Careful analysis over decades of tournament play has proven the usefulness or fairness of the opening sequences. Initial sequences are decided by the opening book until the move is out of the book. Each player chooses its opening randomly and the seven first choices have the same probability to be selected as an opening.

If the first move is 9-13 (Edinburgh), there are 8 openings that start from 9-13. They are Dreaded Edinburgh (9-13, 22-18, 6-9), Edinburgh Single (9-13, 22-18, 11-15), The Garter Snake (9-13, 23-19, 10-15), The Henderson (9-13, 22-18, 10-15), The Inferno (9-13, 22-18, 10-14), The Twilight Zone (9-13, 24-20, 11-16), The Wilderness (9-13, 22-18, 11-16), and The Wilderness II (9-13, 23-18, 11-16). In this case, there are four choices for the second moves: 22-18, 23-19, 24-20, and 23-18. The second move is chosen randomly and the next moves are selected continually in the same manner. Table 2 shows an example of openings in checkers.

Table 2. Openings in Checkers game

Name	Opening	Name	Opening
Edinburgh	9-13	Old Faithful	11-15
Dreaded Edinburgh	9-13, 22-18, 6-9	Cross	11-15, 23-18, 8-11
Edinburgh Single	9-13, 22-18, 11-15	Dyke	11-15, 22-17, 15-19
The Garter Snake	9-13, 23-19, 10-15	Bristol	11-16
The Henderson	9-13,22-18, 10-15	Bristol Cross	11-16, 23-18, 16-20
Double Corner	9-14	Millbury	11-16, 22-18, 8-11
Double Corner Dyke	9-14, 22-17, 11-15, 25-22, 15-19	Oliver's Twister	11-16, 21-17, 8-11
Denny	10-14	Dundee	12-16
Kelso Cross	10-15, 23-18	Bonnie Dundee	12-16, 24-20, 8-12
The Tyne	10-15, 21-17, 9-13	The Skunk	12-16, 24-20, 10-15

Endgame Stage

The estimated quality of the board is calculated using the evolved neural networks to evaluate the leaf nodes of the tree with the minimax algorithm. If the value of f (estimated quality of the next moves) is not reliable, we refer to domain-specific knowledge and revise f. The decision rule for querying the domain knowledge is defined as follows.

IF (f <0.75 and f >0.25) or (f <-0.25 and f >-0.75)
THEN querying the domain knowledge

For example, there is a 2-ply game tree and the concept of selective domain-specific knowledge is like this. In the game tree, let's assume that there are 8 terminals. The two choices are evaluated as 0.3 and -0.9. It is clear that the board configuration as evaluated -0.9 is not good. However, the board configuration with 0.3 is not enough to decide as a draw and needs querying the domain knowledge. If the returned answer from the DB is a draw, the player must select the move. However, if the answer is a loss, the player could select the configuration of -0.9.

3.3 Othello

As mentioned before, we have classified a single Othello game into three stages: opening, middle, and endgame stages. In the opening stage, about 99 previously summarized openings are used to determine the initial moves. In the middle and endgame stage, a weight piece counter is used to search for a good move and an evolutionary algorithm is used to optimize the weights of the counter. Though game tree is useful to find a good move, in this paper we only focus on the goodness of the evaluation function. Figure 5 shows the procedural flow of the proposed method in a game. The longest opening has 18 moves and it can significantly reduce the difficulty of the evaluation function optimization.

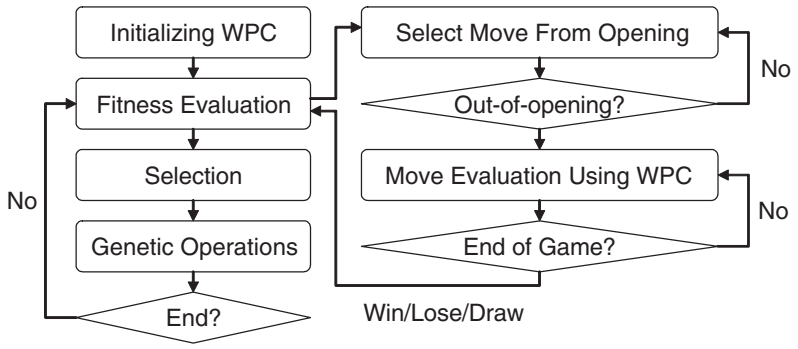


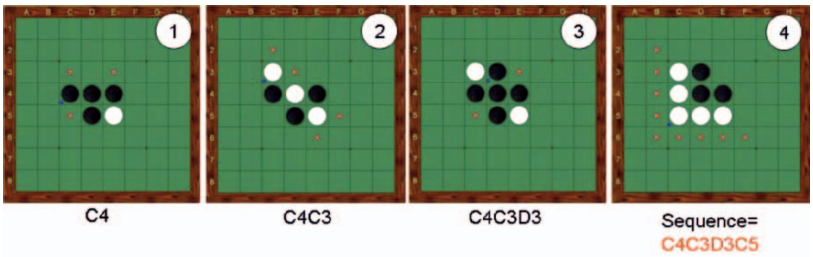
Fig. 5. The flow of the game using the proposed method

Opening Stage

Opening is very important for Othello game because the game is extremely short compared to other games and the importance of early moves is huge. In the games between expert players, gaining small advantage in the early games is very important. They attempt to memorize good or tricky moves before tournament. Brian Rose, World Othello Champion at 2002, said that he memorized about 300 lines before the championships. It is critical to prepare and memorize well-defined openings in the Othello game.

In this work, well-defined 99 openings are used (http://www.oth ello.nl/content/anim/openings.txt). The length of opening ranges from 2 to 18. In fact, in the expert player's game, there is a case that all moves are memorized by both of them. The length of opening can be expanded to the last move, but it is not possible to exploit all of them. "Out-of-opening" means that the moves by the player are not in the list of the openings. Because it is carefully analyzed by experts, it can guarantee the equivalence between two players. In some openings, they are a bit beneficial to a specific color but it is not important because in human's game the difference can be recovered in the middle and end game.

The opening is named after their shapes. Tiger, Cat, and Chimney means that the shape of the opening is similar to the objects. Until out-of-opening, the WPC player follows one of the openings in the list. Figure 6 shows an example of opening selection. The current sequence of the game is defined as $S = \{m_1, m_2, ..., m_n\}$. Here, m_i represents each move, and $n$is the number of moves played. For each opening, we check whether the first n moves are the same with the sequence. The satisfactory openings are called candidates.



Name	Opening	Match	Selection Probability	Selection
X-square	C4C3D3C5B2	O	1/99	X
Snake	C4C3D3C5B3	O	2/99	X
...	...	...	...	...
Cow	C4C3D3C5D6	O	13/99	O
Wing Variation	C4C3E6C5	X	1/99	X
...	...	...	...	...

Fig. 6. The selection of opening (Cow is selected)

Among candidates, one opening is chosen probabilistically. The next move of the selected opening after the first n moves is decided as the next move of player. At the 3^{rd} move, there are 3 choices: Diagonal, perpendicular, parallel. For each choice, there are several extended openings. The probability of choice for each line is determined based on the number of openings. For example, the diagonal opening has 59 variations in his line and about 59% probability to be selected. If there are many variations for each choice, it means that the line is the most preferable one for humans. The probability of selection is based on the human's long investigation.

The probability of opening move selection can be determined using another way. For example, the probability of openings in the games of WTHOR database (It contains all public games played between humans) can be used. Otherwise, specific book evaluation (automatically learned from self-played games of strong program) values can be exploited. WZEBRA (strong Othello program) provides evaluation value for each opening. For example, the X-square opening is -23 (in black's perspective) and Snake is -4.

Evolving Evaluator (WPC)

The evaluation of each WPC (Weighted Piece Counter) is based on the competition against static player. Standard heuristic WPC is used to evaluate the performance of the individual. Figure 7 depicts the standard WPC. Its darkness represents the weight of each position. In Othello, four corners are extremely important and the weight of the corners is 1.0. Other places except the ones near the corner has the similar (0.01∼0.1) weights. The positions near the corners are very dangerous because it gives a chance to the opponent to capture the corner. Because it is static, it cannot evaluate well the dynamics of the relevance of position, but it is still strong compared to other random approaches.

The fitness of the WPC is evaluated based on the 1000 games between the standard WPCs. Given two deterministic strategies, there can be only two games (changing the color). This makes the fitness evaluation difficult and

1.0	–0.25	0.1	0.05	0.05	0.1	–0.25	1.0
0.25	–0.25	0.01	0.01	0.01	0.01	–0.25	–0.25
0.1	0.01	0.05	0.02	0.02	0.05	0.01	0.1
0.05	0.01	0.02	0.01	0.01	0.02	0.01	0.05
0.05	0.01	0.02	0.01	0.01	0.02	0.01	0.05
0.1	0.01	0.05	0.02	0.02	0.05	0.01	0.1
0.25	–0.25	0.01	0.01	0.01	0.01	–0.25	–0.25
1.0	–0.25	0.1	0.05	0.05	0.1	–0.25	1.0

Fig. 7. Weights of the heuristic

random moves are used. 10% of moves of both players are decided randomly. It allows 1000 games among them being different but it also makes the fitness of each WPC unstable. To preserve good solutions for the next generation, elitist approach is used.

If the number of wins is W, the number of loses is L, and the number of draws is D, the fitness is defined as follows.

$$fitness = W * 1.0 + D * 0.5$$

The weights are widely used in real Othello tournament.

The weights of each position are initialized as a value between -1 to 1. Until out-of-opening, the WPC in the population uses opening knowledge. After out-of-opening, the WPC evaluates the relevance of board and decides the next move. Among possible moves, the one with the highest WPC is selected. Roulette-wheel selection is used and 1-point crossover is applied to the converted 1-dimensonal array of the 8×8 board. Mutation operator changes an element of the vector as a new value ranging from -1 to 1.

4 Experimental Results

4.1 Checkers

Speciation algorithm is used to generate diverse strategies and improve the performance of evolutionary strategies by combining multiple strategies [43]. The non-speciated evolutionary algorithm uses a population size of 100 and limits the run to 50 generations. The speciated evolutionary algorithm sets the population size to 100, generations to 50, the mutation rate to 0.01 and crossover rate to 1.0. The number of leagues (used to select the best player from each species) is 5 (5 means that each player selects 5 players from the species randomly and the competition results are used for the selection). Evolving checkers using speciation requires 10 hours on a Pentium III 800MHz (256MB RAM). The non-speciated evolutionary algorithm uses only mutation but the speciated evolutionary algorithm uses crossover and mutation. The non-speciated evolutionary algorithm is the same as Chellapilla and Fogel's checkers program. The parameters of a simple EA are Population_Size=100, Mutation_rate=1.0, Generation=50. The parameters of the speciated EA are Population_Size=100, Mutation_rate=0.01, Crossover_rate=1.0 and Generation=50. They have the same number of individuals for evolution and the game that they played for one generation is the same to ours.

The Chinook endgame database (2∼6 pieces) is used for revision when the estimated value from the neural network is between 0.25 and 0.75 or between -0.25 and -0.75. Time analysis indicates that the evolution with knowledge takes much less time than that without knowledge in simple evolution (Figure 8) and the knowledge-based evolution takes a little more time than that without knowledge in the speciated evolution (Figure 8). This means that

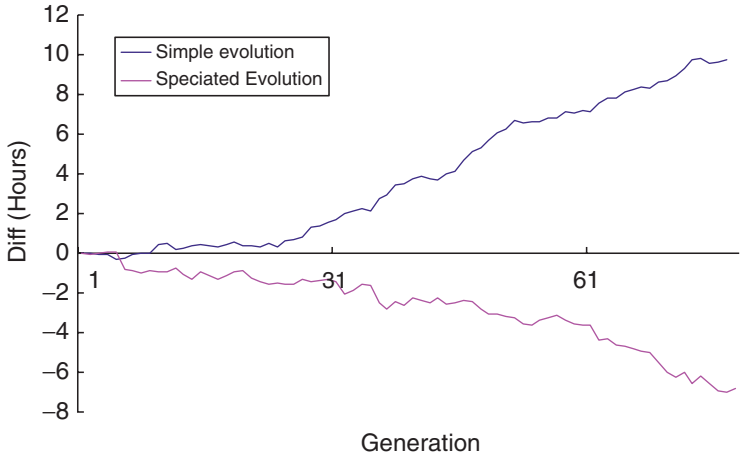


Fig. 8. Comparison of running time (Diff = Time(Without Knowledge)-Time(With Knowledge))

Table 3. The competition results between the speciated players using both opening and endgame DB and the speciated player with one of the knowledge (Win/Lose/Draw)

Speciated EA with opening and endgame DB	Speciated EA with opening DB	14/6/10
Speciated EA with opening and endgame DB	Speciated EA with endgame DB	9/10/11
Speciated EA with opening DB	Speciated EA with endgame DB	5/13/12

the insertion of knowledge within a limited scope can accelerate the speed of the evolutionary algorithm because it can reduce the computational requirement for finding an optimal endgame sequence by using the endgame DB. Since we have used two different machines to evolve simple and speciated versions, respectively, direct comparison of evolution time between them is meaningless. In the speciated evolution, the insertion of knowledge increases the evolution time and an additional 5 hours are needed for 80 generations. Table 3 shows the effect of the stored knowledge (opening and endgame databases) in speciation.

4.2 Othello

The parameters of the evolution are as follows. The maximum generation is 100, population size is 50, crossover rate is 0.8, and mutation rate is 0.01. The best fitness is about 540 in the evolution of WPC with opening knowledge. After then, it converges to 500 though the population size is 50. The best individual at the 1^{st} generation is 400. It means that the incorporation of the

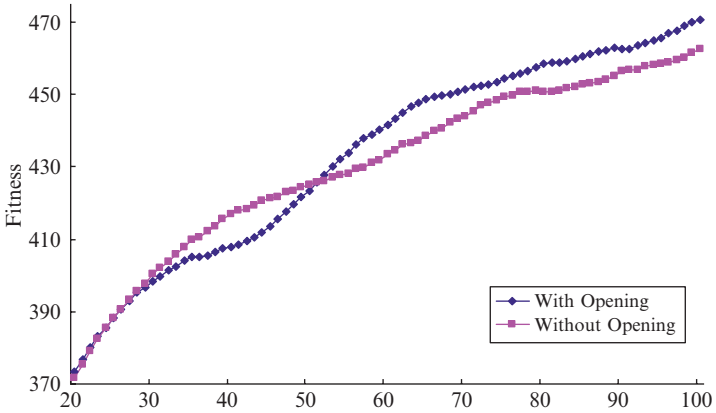


Fig. 9. Average of 10 runs. The value is average of 20 generations to show trends

opening knowledge gives some individuals the high performance gain. The average fitness at the 1^{st} generation is about 200. It is not quite different with the one of evolution without opening. That means the incorporation of opening does not increase the performance of all individuals in the population.

The evolution finds solutions that can do well after the opening and it can significantly reduce the burden for the WPC. Though the generations from 21 to 45 show performance degradation, the best individual that exploits the advantage of opening emerges at 51. The comparison of the maximum fitness over the evolution between the one with opening and the one without opening shows that the one with opening shows better fitness during evolution. Though the competition dose not support opening and the direct comparison against them is not possible, the 550 is about rank 12 in the CEC competition in 2006 [44].

The comparison of the average fitness between them shows that the average fitness of the one with opening is larger than that without opening, but it finally converges to the same value. It means the incorporation of opening does not give huge advantage to the population. The average fitness of the 1^{st} generation is not different and it means that the evolution finds more improved individuals that exploit the opening knowledge. Figure 9 shows the average of 10 runs.

5 Concluding Remarks

In this chapter, we presented several methods to incorporate domain knowledge into evolutionary board game players: checkers and Othello. Well-defined opening and endgame knowledge can be easily incorporated into the evolutionary induction procedure of strategies and experimental results confirm

that the presented methods can improve the performance of evolved strategies compared to that without knowledge.

Acknowledgements

This research was supported by Brain Science and Engineering Research Program sponsored by Korean Ministry of Commerce, Industry and Energy.

References

1. Schaeffer, J.: One Jump Ahead: Challenging Human Supremacy in Checkers. Springer (1997)
2. Fogel, D.B.: Blondie 24: Playing at the edge of AI. Morgan Kaufmann (2001)
3. Kim, K.-J. and Cho, S.-B.: Systematically incorporating domain-specific knowledge into evolutionary speciated checkers players. IEEE Transactions on Evolutionary Computation, 9(6) (2005) 615-627
4. Kim, K.-J. and Cho, S.-B.: Evolutionary Othello players boosted by opening knowledge. World Congress on Computational Intelligence (2006)
5. Fogel, D.B., Hays, T.J., Hahn, S. and Quon, J.: A self-learning evolutionary chess program. Proc. of the IEEE 92(12) (2004) 1947-1954
6. Chernev, I.: The Compleat Draughts Player. Oxford University Press, (1981)
7. Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P. and Szafron, D.: A world championship caliber checkers program. Artificial Intelligence 53(2-3) (1992) 273-290
8. Schaeffer, J., Lake, R., Lu, P. and Bryant, M.: Chinook: The man-machine world checkers champion. AI Magazine 17(1) (1996) 21-29
9. R. Lake, J. Schaeffer and P. Lu, "Solving large retrograde analysis problems using a network of workstations," *Proc. of Advances in Computer Chess VII*, pp. 135-162, 1994.
10. Fogel, D.B.: Evolutionary entertainment with intelligent agents. IEEE Computer, 36(6) (2003) 106-108
11. Chellapilla, K. and Fogel, D.B.: Evolving neural networks to play checkers without relying on expert knowledge. IEEE Trans. on Neural Networks 10(6) (1999) 1382-1391
12. Chellapilla, K. and Fogel, D.B.: Evolving an expert checkers playing program without using human expertise. IEEE Trans. on Evolutionary Computation 5(4) (2001) 422-428
13. Fogel, D.B.: Evolving a checkers player without relying on human experience. ACM Intelligence, 11(2) (2000) 20-27
14. Chellapilla, K. and Fogel, D.B.: Anaconda defeats Hoyle 6-0: A case study competing an evolved checkers program against commercially available software. Proc. of the 2000 Congress on Evolutionary Computation 2 (2000) 857-863
15. Fogel, D.B. and Chellapilla, K.: Verifying Anaconda's expert rating by competing against Chinook: Experiments in co-evolving a neural checkers player. Neurocomputing 42(1-4) (2002) 69-86

16. Chong, S.Y., Tan, M.K. and White, J.D.: Observing the evolution of neural networks learning to play the game of Othello. IEEE Transactions on Evolutionary Computation 9(3) (2005) 240-251
17. Sun, C.-T. and Wu, M.-D.: Multi-stage genetic algorithm learning in game playing. NAFIPS/IFIS/NASA '94 (1994) 223-227
18. Sun, C.-T. and Wu, M.-D.: Self-adaptive genetic algorithm learning in game playing. IEEE International Conference on Evolutionary Computation 2 (1995) 814-818
19. Moriarty, D.E. and Miikkulainen, R.: Discovering complex Othello strategies through evolutionary neural networks. Connection Science 7 (1995) 195-209
20. Moriarty, D. E. and Miikkulainen, R.: Evolving neural networks to focus minimax search. Proc. of the 12^{th} National Conf. on Artificial Intelligence (AAAI-94) (1994) 1371-1377
21. Moriarty, D.E. and Miikkulainen, R.: Improving game-tree search with evolutionary neural networks. Proc. of the First IEEE Conf. on Evolutionary Computation, 1 (1994) 496-501
22. Richards, N., Moriarty, D. and Miikkulainen, R.: Evolving neural networks to play go. Applied Intelligence 8 (1998) 85-96
23. Stanley, K.O. and Miikkulainen, R.: Evolving a roving eye for go. Proc. of the Genetic and Evolutionary Computation Conference (GECCO-2004) (2004) 1226-1238
24. Lubberts, A. and Miikkulainen, R.: Co-evolving a go-playing neural networks. Proc. of 2001 Genetic and Evolutionary Computation Conference Workshop Program (GECCO-2001), (2001) 14-19
25. Perez-Bergquist, A.S.: Applying ESP and region specialists to neuro-evolution for go. Technical Report CSTR01-24, Department of Computer Science, The University of Texas at Austin, May (2001)
26. Barone, L. and While, L.:An adaptive learning model for simplified poker using evolutionary algorithms. Proc. of the 1999 Congress on Evolutionary Computation 1 (1999) 153-160
27. Pollack, J.B. and Blair, A.D.: Co-evolution in the successful learning of backgammon strategy. Machine Learning 32(3) (1998) 225-240
28. Kendall, G. and Whitwell, G.: An evolutionary approach for the tuning of a chess evaluation function using population dynamics. Proc. of the 2001 Congress on Evolutionary Computation 2 (2001) 995-1002
29. Kendall, G. and Smith, C.: The evolution of blackjack strategies. Proc. of the 2003 Congress on Evolutionary Computation 4 (2003) 2474-2481
30. Fogel, D.B.: Evolving strategies in blackjack. Proc. of the 2004 Congress on Evolutionary Computation (2004) 1427-1434
31. Ono, W. and Lim, Y.-J.: An investigation on piece differential information in co-evolution on games using Kalah. Proc. of Congress on Evolutionary Computation 3 (2003) 1632-1638
32. Lucas, S.M. and Kendall, G.: Evolutionary computation and Games. IEEE Computational Intelligence Magazine (2006) 10-18
33. Chong, S.Y., Ku, D.C., Lim, H.S., Tan, M.K. and White, J.D.: Evolved neural networks learning Othello strategies. The 2003 Congress on Evolutionary Computation 3 (2003) 2222-2229
34. Sun, C.-T., Liao, Y.-H., Lu, J.-Y. and Zheng, F.-M.: Genetic algorithm learning in game playing with multiple coaches. IEEE World Congress on Computational Intelligence 1 (1994) 239-243

35. Alliot, J. and Durand, N.: A genetic algorithm to improve an Othello program. Artificial Evolution (1995) 307-319
36. Eskin, E. and Siegel, E.: Genetic programming applied to Othello: Introducing students to machine learning research. Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education, 31(1) (1999) 242-246
37. Bateman, R.: Training a multi-layer feedforward neural network to play Othello using the backpropagation algorithm and reinforcement learning. Journal of Computing Sciences in Colleges 19(5) (2004) 296-297
38. Smith, R.E. and Gray, B.: Co-adaptive genetic algorithms: An example in Othello Strategy. TCGA Report No. 94002, The University of Alabama (1994)
39. Leuski, A. and Utgoff, P.E.: What a neural network can learn about Othello. Technical Report TR96-10, Department of Computer Science, University of Massachusetts, Amherst (1996)
40. Leuski, A.: Learning of position evaluation in the game of Othello. Technical Report TR 95-23, Department of Computer Science, University of Massachusetts, Amherst (1995)
41. Moriarty, D.E., Schultz, A.C. and Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. Journal of Artificial Intelligence Research 11 (1999) 241-276
42. Hansen, L.K., and Salamon, P.: Neural network ensembles. IEEE Trans. on Pattern Analysis and Machine Intelligence 12(10) (1990) 993-1001
43. Kim, K.-J. and Cho, S.-B.: Evolving speciated checker players with crowding algorithm. Congress on Evolutionary Computation (2002) 407-412
44. CEC 2006 Othello Competition, http://algoval.essex.ac.uk:8080/othello/html/Othello.html.