Systematically Incorporating Domain-Specific Knowledge Into Evolutionary Speciated Checkers Players

Kyung-Joong Kim, *Student Member, IEEE,* and Sung-Bae Cho, *Member, IEEE*

***Abstract*—The evolutionary approach for gaming is different from the traditional one that exploits knowledge of the opening, middle, and endgame stages. It is, therefore, sometimes inefficient to evolve simple heuristics that may be created easily by humans because it is based purely on a bottom-up style of construction. Incorporating domain knowledge into evolutionary computation can improve the performance of evolved strategies and accelerate the speed of evolution by reducing the search space. In this paper, we propose the systematic insertion of opening knowledge and an endgame database into the framework of evolutionary checkers. Also, the common knowledge that the combination of diverse strategies is better than a single best one is included in the middle stage and is implemented using crowding algorithm and a strategy combination scheme. Experimental results show that the proposed method is promising for generating better strategies.**

***Index Terms*—Checkers, combination, domain knowledge, endgame database, opening knowledge, speciation.**

I. Introduction

SINCE THE beginning of the computer age, people have been eager to create an intelligent game program capable of defeating human experts. Many different approaches have been used for different games including neural networks for backgammon [1], special-purpose hardware called Deep Blue for chess [2], and the application of expert knowledge with relatively small computational power for checkers [3] and Othello [4]. Most of these techniques exploit expert knowledge as much as possible, such as the proper learning algorithm for training the evaluation function, relevance factors for the evaluation, the weights of the evaluation factors, opening knowledge, and an endgame database. Acquiring such knowledge requires the help and advice of game experts, computational power for processing the knowledge extracted, and a process of trial and error to find the best overall approach. By using many programmers and players, expert knowledge can be digitalized and be made accessible through the Internet.

Traditional methods for developing strategies for games such as checkers and chess divide the game into opening, middle, and endgame stages. For each stage, a different heuristic is applied. For example, it is very difficult to determine the most appropriate choice in the opening stage of a game, so the use of an opening book from games played by experts is beneficial. In the middle stage, a game tree with a limited depth is constructed and a heuristic evaluation function is applied to estimate the relevance of each move. Finally, in the end game, the number of pieces (or possible moves) becomes reasonably small and deterministic calculation of the final moves is possible.

In checkers, the typical approach uses a computer program to search a game tree to find an optimal move at each play, but there are challenges in overcoming an expert's experience in the opening, middle, and endgame stages. Sometimes a computer checkers program fails to defeat a human player because it makes a mistake that is not common among human players. Sometimes the fault is discovered by the computer program after searching beyond the predefined depth of the game tree (a so-called "horizon effect"). To defeat the best human players, Chinook (the best computer checkers program) uses an opening book, and most importantly, the endgame database. Also, Chinook relies on expert knowledge that is captured in an evaluation function, which is used in the middle stage. Chinook's success is based largely on traditional game theory mechanics (game tree and alpha-beta search) and expert knowledge (opening book, components in evaluation function, and endgame database).

Recently, evolutionary induction of game strategies has gained popularity because of the success reported in [5] using the game of checkers. This approach does not need additional prior knowledge or expert heuristics for evolving strategies and expert-level strategies have evolved from the process of self-play, variation, and selection. In other games such as Othello [6], [7], blackjack [8], Go [9], chess [10], and backgammon [11], the evolutionary approach has been applied to discover better strategies without relying on human experience. Usually, opening knowledge and endgame databases are not involved in the evolutionary process because a researcher wants to investigate the possibility of pure evolution [5]. However, it might take a long evolution time to create a world-level champion program without a predefined knowledge base. It took six months (using a Pentium II machine) to evolve the checkers program rated at the expert level by Fogel and Chellapilla [25], and it would take even longer time to evolve the world-level champion program.

Incorporating *a priori* knowledge, such as expert knowledge, metaheuristics, human preferences, and most importantly, domain knowledge discovered during evolutionary search, into evolutionary algorithms (EAs) has gained increasing interest in recent years [12]. In this paper, we propose a method for systematically inserting expert knowledge into evolutionary checkers

Manuscript received August 29, 2004; revised January 17, 2005 and May 22, 2005. This work was supported in part by the Brain Science and Engineering Research Program sponsored by Korean Ministry of Commerce, Industry, and Energy.

The authors are with the Department of Computer Science, Yonsei University, Seoul 120-749, Korea (e-mail: kjkim@cs.yonsei.ac.kr; sbcho@cs.yonsei.ac.kr).

Digital Object Identifier 10.1109/TEVC.2005.856213

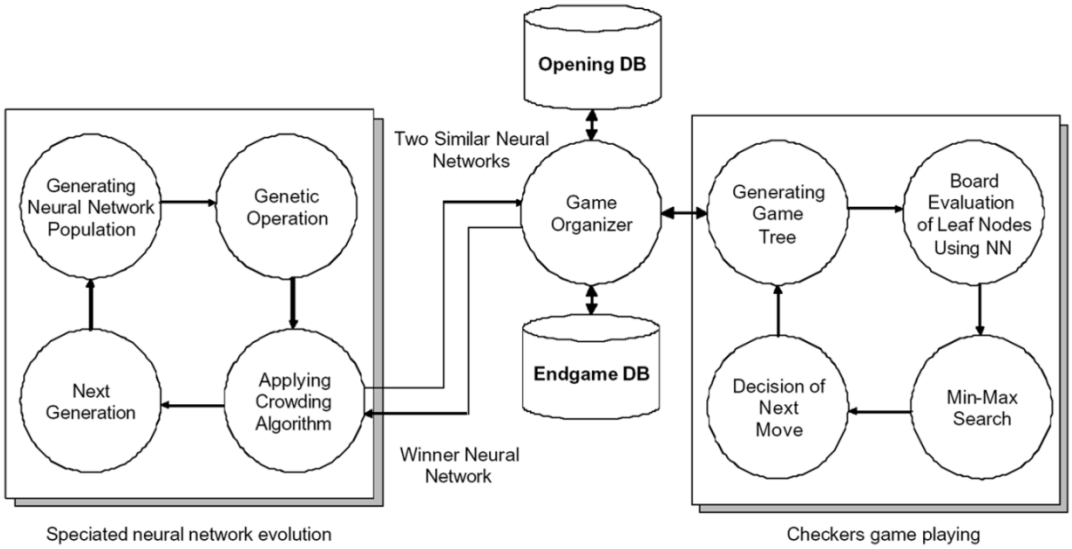


Fig. 1. Conceptual diagram of the proposed method. Game organizer decides the usage of the opening DB, game tree, and the endgame DB.

framework [5] at the opening, middle, and endgame stages. In the opening stage, openings defined by the American Checkers Federation (ACF) are used. In previous work, we have used speciation techniques to search for diverse strategies that embody different styles of game play and have combined them using voting for higher performance [13], [14]. 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 have used an endgame database from Chinook, the first man-machine checkers champion. Fig. 1 explains the conceptual framework of the proposed method.

The idea of this paper is the systematic incorporation of three knowledge domains (opening database (DB), middle stage knowledge, and endgame DB) into an evolutionary checkers players with a speciation algorithm in the middle stage and a predefined rule for using an endgame DB. The middle stage knowledge comes from the Korean event in the game of Go. In 2003, the Internet site TYGEM (http://www.tygem.co.kr) held a many-to-one style game between Hoon Hyun Cho, one of the greatest Go players, and 3000 amateur players. The winner of the game was Cho. After the game, he said that it was a very difficult game because the amateur players did not make any obvious mistake. A speciation algorithm is used to simulate the effect of many amateur players in our evolutionary checkers program.

The rest of this paper is organized as follows. Section II describes the background including the research on evolution and games. Section III applies speciation to the evolution process and presents the knowledge insertion method. Section IV describes the experimental results and analysis.

II. Background

A. Checkers

Checkers is traditionally played on an eight-by-eight board (see Fig. 2) and there are two players ("red" and "white"). Each player has 12 pieces (checkers) and the red player moves first. Checkers are allowed to move forward diagonally one square at a time. In the case that a jump condition is satisfied, one can jump diagonally over an opposing checker and the opposing checker is removed. Jumps are mandatory. When a checker advances to the last row of the board, it becomes a king and can move diagonally one square at a time in any direction. The game ends when a player has no more available moves (that player without moves is the loser) and the game can also end when one side offers a draw and the other accepts.

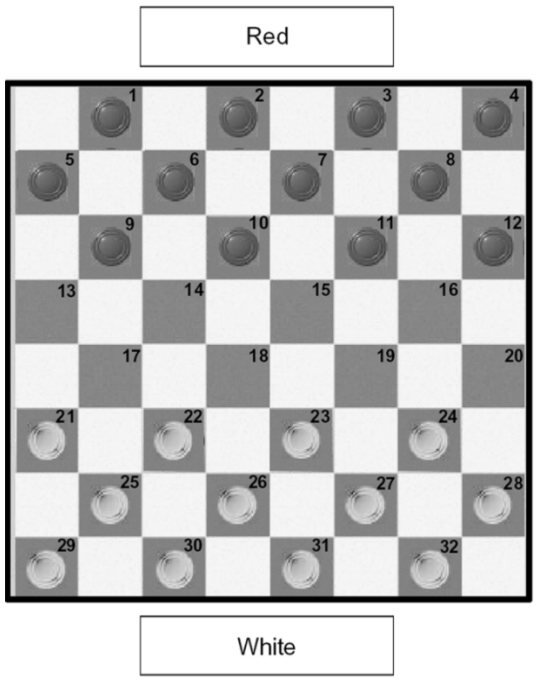


Fig. 2. Opening board in a checkers game. Red moves first.

Checkers has a smaller search space than chess, approximately 5×10^{20} positions versus $O(10^{44})$ for chess [36]. The search space is small enough that one could consider solving the problem. Chinook, one of the best checkers program, has not solved the game but it is playing at a level that makes it almost unbeatable. The average number of moves to consider

in a checkers position (called the branching factor) is less than that for chess. A typical checkers position without any captures has eight legal moves (for chess it may be 35–40). As a result, checkers programs can search deeper than their chess counterparts. Checkers provides an easier domain to work with, and provides the same basic research opportunities as does chess or Go.

B. Traditional Game Programming

A game can usually 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 [15]. A problem with this approach is that the program will follow published play, which is usually familiar to the humans [16]. 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.

One of the 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 requires 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 nets, and EAs and can compete with hand-tuning [17], [18].

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 eight 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 2.7 GB (compressed) [19]. Recently, the construction of a ten-piece database has been completed.

C. Evolution and Games

Checkers is the board game for which evolutionary computation has been used to evolve strategies. Fogel *et al.* have explored the potential for a coevolutionary 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 playing well [20]–[23]. 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 [24]. A series of ten games against a "novice-level" version of Chinook, a high-level expert, resulted in two wins, four losses, and four draws [25].

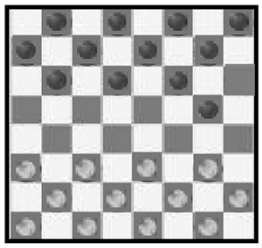
TABLE I
SUMMARIZATION OF RELATED WORKS IN EVOLUTIONARY GAMES

Information	Game	Reference	Game Tree	Neural Network	Co-evolution	Knowledge Incorporation
Perfect Information	Checkers	[5]	O	O	O	
	Othello	[6]	O	O	O	
		[7]	O		O	O
		[26]		O		
		[27]	O	O		
	Go	[9]		O	O	
	Chess	[10]	O		O	O
		[18]	O	O	O	O
	Kalah	[34]	O	O	O	O
Imperfect Information	Backgammon	[11]		O	O	
	Poker	[32]				O
	Blackjack	[8]		O	O	O
		[33]				O

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 [6]. 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 [7]. Moriarty *et al.* designed an evolutionary neural network that output the quality of each possible move at the current board configuration [26]. Moriarty *et al.* evolved neural networks to constrain minimax search in the game of Othello [27], [28]. At each level, the network saw the updated board and the rank of each move and only a subset of these moves was explored.

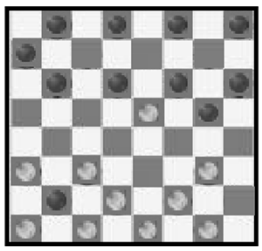
The symbiotic, adaptive neuroevolution (SANE) method was used to evolve neural networks to play the game of Go on small boards with no preprogrammed knowledge [9]. Stanley *et al.* evolved a roving eye neural network for Go to scale up by learning on incrementally larger boards, each time building on knowledge acquired on the prior board [29]. 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 coevolutionary techniques of competitive fitness sharing, shared sampling, and a hall of fame to the SANE neuroevolution method [30]. Santiago *et al.* applied an enforced subpopulation variant of SANE to Go and an alternate network architecture featuring subnetworks specialized for certain board regions [31].

Barone *et al.* used EAs to learn to play games of imperfect information—in particular, the game of poker [32]. They identified several important principles of poker play and used them

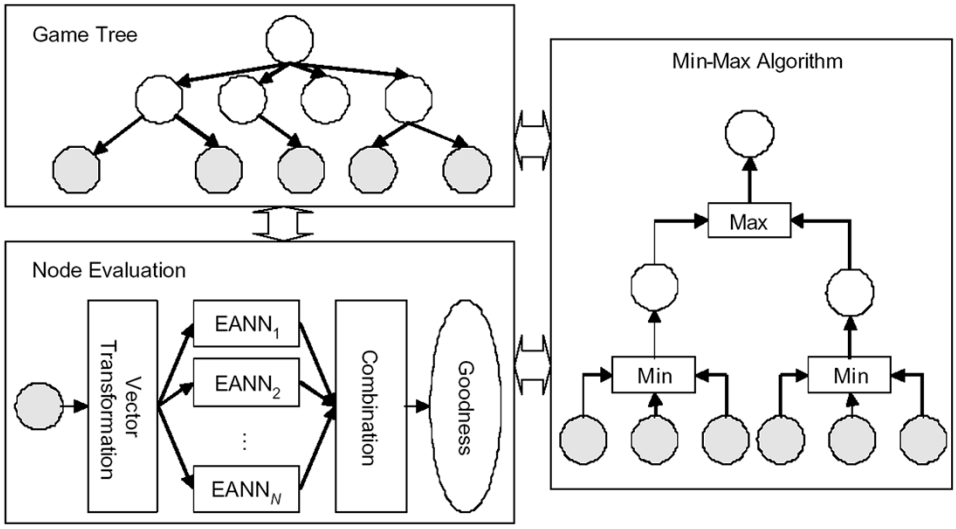


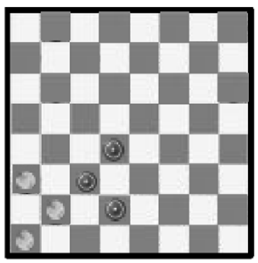
Opening Stage

Name	Opening	Name	Opening
Dreaded Edinburgh	9-13, 22-18, 6-9	The Black Hole	10-14, 22-17, 9-13
Edinburgh Single	9-13, 22-18, 11-15	...	...
...	...	Boston	11-15, 22-17, 9-13, 17-14
Double Cross	9-14, 23-18, 14-23	...	...
Pluto	9-14, 22-18, 14-17	Paisley	11-16, 24-19, 8-11, 22-18
...	...	...	...



Middle Stage





Endgame Stage

Positions	Turn	Number of Pieces	Result
Bowen's Twins	Black	8	Win
...	...	...	...
Johnson's Position	White	6	Win
...	...	...	...
Payne's Draw	White	5	Draw
Roger's Draw	White	4	Draw

Fig. 3. Flow of the game using the proposed method. Each evolutionary artificial neural network (EANN) evaluates the terminal nodes of the game tree and the voting of the neural networks determines the final move.

as the basis for a hypercube of evolving populations. Coevolutionary learning was used in backgammon [11] and chess [10], [18]. Kendall *et al.* utilized three neural networks (one for splitting, one for doubling down, and one for standing/hitting) to evolve blackjack strategies [8]. 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 [33]. Ono *et al.* utilized coevolution of artificial neural networks on a game 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) [34]. Table I summarizes the related works. Fogel's checkers framework has been used in other games such as [6], [8], [18], and [34].

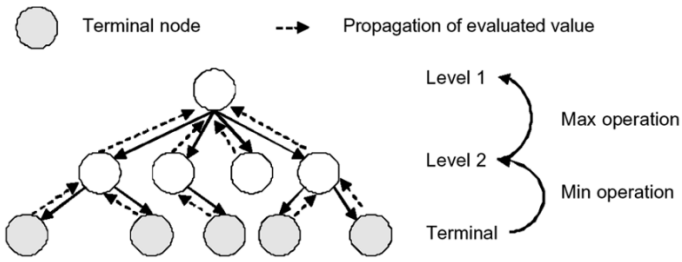


Fig. 4. Upward propagation of the evaluated value through game tree using min/max operations.

Fogel *et al.* applied the framework to the game of chess and reported that the evolved program performed above the master level [18].

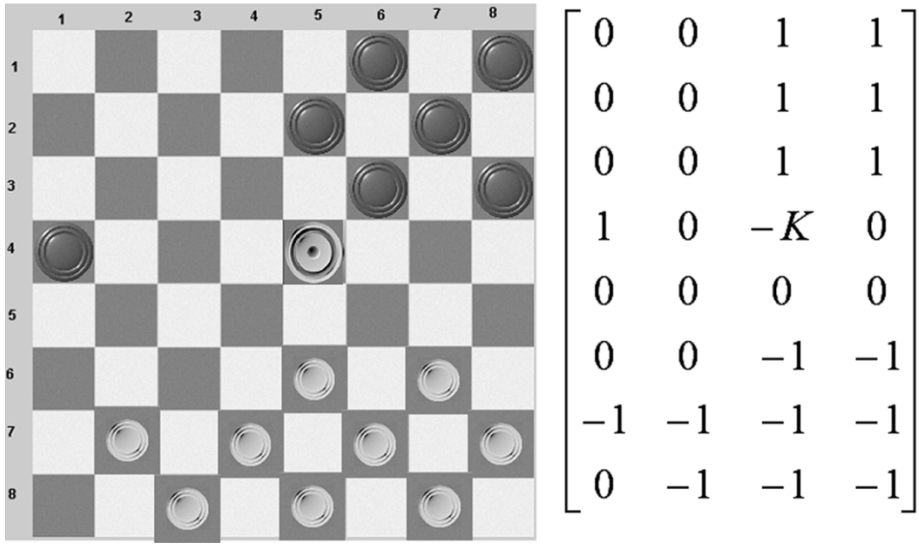


Fig. 5. Example of board representation. Minus means the opponent checkers and K means the King. The value of K is evolved with the neural networks.

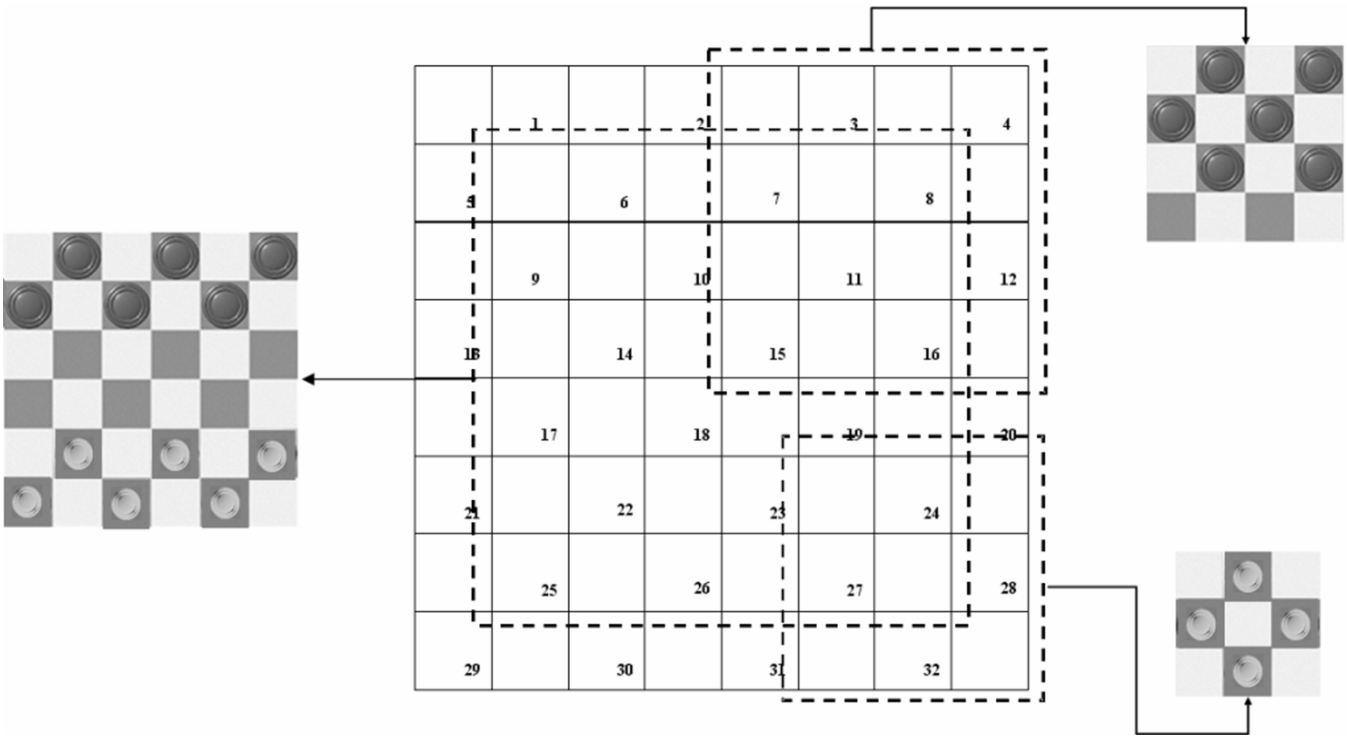


Fig. 6. Example of 6×6, 4×4, and 3×3 subboards. In a checkerboard, there are 91 subboards (36 3×3 subboards, 25 4×4 subboards, 16 5×5 subboards, 9 6×6 subboards, 4 7×7 subboards, and 1 8×8 subboards). This design provides spatial local information to the neural networks.

III. Incorporating Knowledge Into Evolutionary Checkers

As mentioned before, we have classified 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 [35]. Because the fitness landscape of an evolutionary game evaluation function is highly dynamic, a speciation technique such as fitness sharing is not appropriate. A crowding algorithm that can cope with a dynamic landscape is adopted for generating more diverse neural network evaluation functions. The performance of evolutionary neural networks for creating a checkers evaluation function has been demonstrated by many researchers [5]. 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). Fig. 3 shows the procedural flow of the proposed method in a game.

A. 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 in Fig. 2 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 eight 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.

B. Evolutionary Checkers

1) Concept of a Game Tree: To find the next move of a player, a game tree is constructed with a limited depth. Each node in a game tree represents the configuration of the board at some stage of the game. The quality of the terminal nodes is measured with the evaluator. The evaluated values of the terminal nodes propagate upward using min/max operations. The max operation chooses the max value of all children nodes and the min operation chooses the min value. The current configuration of the board is represented as a root node and the arc represents a move. At an odd number level, the max operation is used and *vice versa*. Fig. 4 describes the procedure for a two-ply example.

2) Evaluation of a Board Configuration: Usually, an evaluation function is the linear sum of the values of relevant features selected by experts. The 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 requires 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.

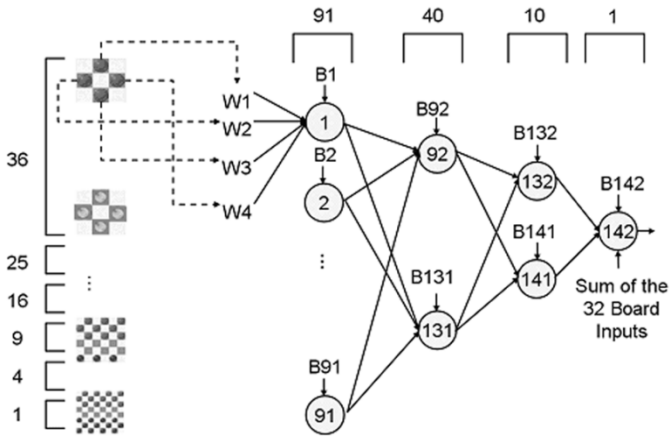


Fig. 7. Architecture of neural network. It is fully connected in the hidden layers. One subboard is transformed into the corresponding vector representation and used for the input of the neuron. The number of neurons is followed from [5]. The output of the neural network indicates the quality of the board configuration.

3) Neural Networks for Evaluation: The feed-forward neural network, which has three hidden layers comprising 91 nodes, 40 nodes, and 10 nodes, respectively, is used as an evaluator. The board configuration is an input to the neural network that evaluates the configuration and produces 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. Following Fogel [5], 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. Fig. 5 describes the representation of board.

For reflecting spatial features of the board configuration, subboards of the board are used as an input. One board can have 36 3×3 subboards, 25 4×4 subboards, 16 5×5 subboards, 9 6×6 subboards, 4 7×7 subboards, and 1 8×8 subboard. 91 subboards are used as an input to the feed-forward neural network. Fig. 6 shows an example of 3×3, 4×4, and 6×6 subboards. 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. Fig. 7 describes the architecture of the neural network.

4) Details of Evolutionary Search: 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. Fig. 8 describes the structure of the chromosome. An offspring $P_i', i = 1, \ldots, p$ for each parent $P_i, i = 1, \ldots, p$ is created by

$$\sigma_i'(j) = \sigma_i(j) \exp(\tau N_j(0,1)), \quad j = 1, \ldots, N_w$$
$$w_i'(j) = w_i(j) + \sigma_i'(j) N_j(0,1), \quad j = 1, \ldots, N_w$$

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

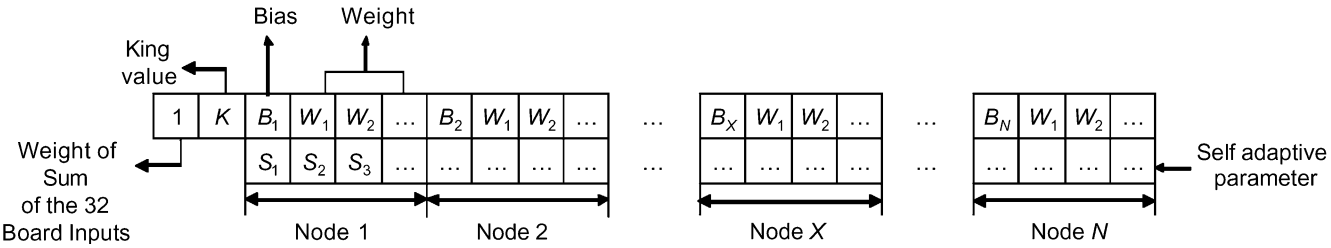


Fig. 8. Structure of the chromosome (N is the number of nodes). Each node (142 nodes in total) has bias values and the weights of the input signals.

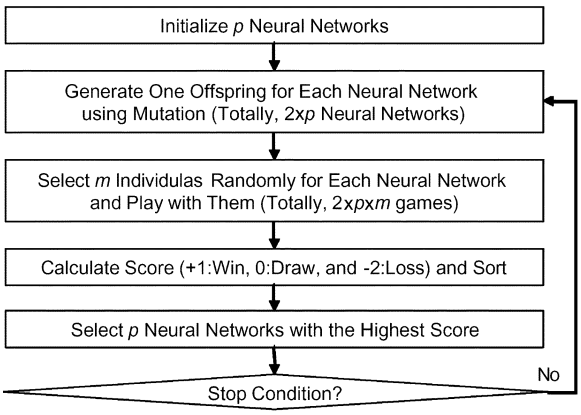


Fig. 9. Evolutionary procedure for checkers (p is the number of individuals. m is the number of opponents to play with). If the number of generations is larger than the previously defined maximum, the procedure stops.

$N_j(0,1)$ 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. Fig. 9 summarizes the evolutionary procedure for checkers.

C. Speciated Evolutionary Checkers

An ensemble of evolutionary neural networks can perform better than the single best one [37]. In the Go community, on-line matches between a professional player and a number of amateur players are interesting events. Each move of the amateur players is decided by voting. It is natural for a professional player to defeat an amateur player in a one-to-one match. However, the combination of opinions of multiple players can be as powerful as single professional player. Fig. 10 describes the idea and the implementation in evolutionary checkers. The goal of this approach is to improve the performance in middle stage.

In this paper, we utilize a crowding algorithm [38], a popular form of speciation algorithm, to search for diverse neural networks. In this algorithm, one neural network is selected from

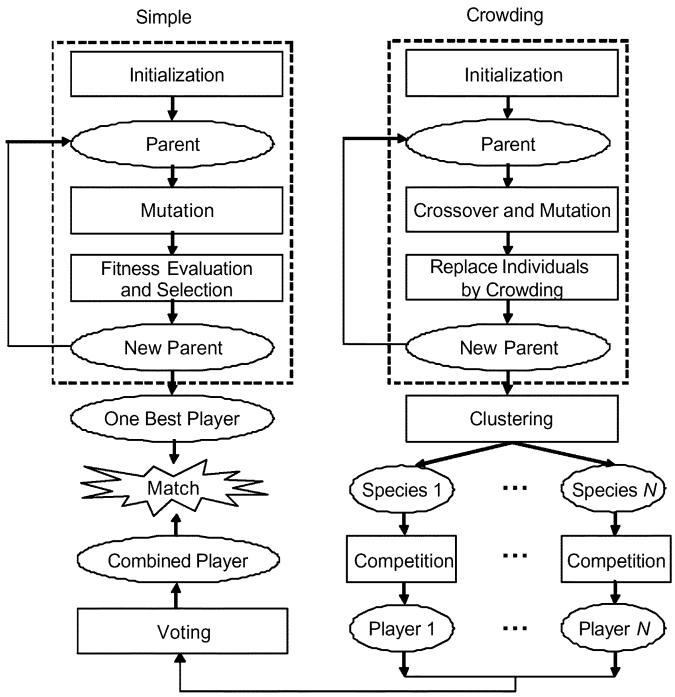


Fig. 10. Evolutionary process for speciated checkers. A crowding algorithm is used for the speciation of population and it is easily implemented. A clustering algorithm is used for the identification of clusters in the population. A tournament-style league is conducted to select the best player in the cluster.

two similar individuals based on the result of game played between them (usually, a crowding algorithm uses their fitness but in this case, we cannot use fitness because of the dynamic property of fitness landscape). A crowding algorithm is one of the representative speciation methods that attempt to discover diverse species in a search space. Fig. 11 shows the algorithm of crowding procedure. The distance between two neural networks is calculated by using the Euclidean distance between their weights. To discover clusters with arbitrary shape, density-based clustering methods have been used. Density-based spatial clustering of applications with noise (DBSCAN) is one of the algorithms [39]. The algorithm is described in Fig. 12. Moves of combined players are determined using a simple voting mechanism. It picks the move that has the greatest number of votes. If there is no clear winner, one of the moves that has the greatest votes is selected randomly. Fig. 13 summarizes the evolutionary process using crowding and the combination of strategies.

D. Endgame Stage

The estimated quality of the board is calculated using the evolved neural networks to evaluate the leaf nodes of the tree

```
1: Initialize P individuals; /* each individual represents a neural network evaluator */
2: Insert P individuals into Q; /*Q holds individuals of the population*/
3: for (i = 0 ; i < MAX_GEN; i++)
4:     Shuffle P individuals in Q;
5:     while (Q is not empty) do
6:         Delete (p1, p2) from Q;
7:         q1, q2=Crossover p1, p2;
8:         r1, r2=Mutate q1, q2;
9:         if (distance (p1, r1)+distance (p2, r2) < distance (p1, r2)+distance (p2, r1)) then
10:            if (fitness (p1) < fitness (r1)) then Insert (r1) to Q2; else Insert (p1) to Q2; end if
11:            if (fitness (p2) < fitness (r2)) then Insert (r2) to Q2else Insert (p2) to Q2; end if
12:        else
13:            if (fitness (p1) < fitness (r2)) then Insert (r2) to Q2 else Insert (p1) to Q2; end if
14:            if (fitness (p2) < fitness (r1)) then Insert (r1) to Q2 else Insert (p2) to Q2; end if
15:        end if
16:    end while
17:    Q=Q2;
18: end for
```

Fig. 11. Pseudocode for crowding algorithm.

```
1: /* P is the size of a population */
2: /* MinPts is the minimum number of individuals to form a new cluster */
3: /* ε is the radius of neighborhood */
4: Insert P individuals into Q; /* Q holds individuals of the population */
5: while (Q is not empty) do
6:    Delete (p) from Q;
7:    if (neighbor_count> MinPts) then
8:       /* neighbor_count is the number of neighbors of p */
9:       C=New_Cluster(p);
10:         if (neighbor of p is included in Cluster Cr) then
11:            Merge (C, Cr);
12:         end if
13:   end if
14: end while
```

Fig. 12. Pseudocode for DBSCAN.

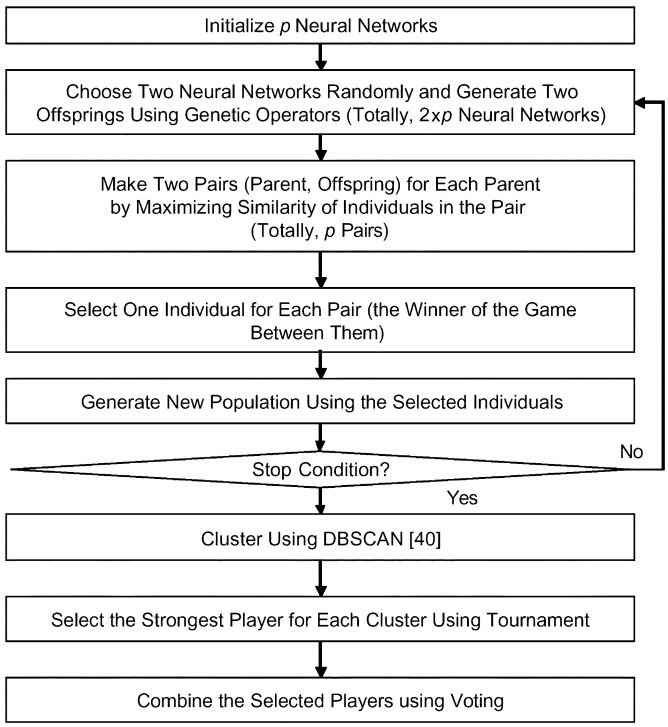


Fig. 13. Evolutionary procedure for checkers (p is the number of individuals). If the number of generations is larger than the maximum previously defined, the procedure stops.

with the min–max 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.

Fig. 14 shows a two-ply game tree and the concept of selective domain-specific knowledge. In this game tree, there are eight 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.

IV. Experimental Results

The nonspeciated EA uses a population size of 100 and limits the run to 50 generations. The speciated EA 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 five (five means that each player selects five players from the species randomly and

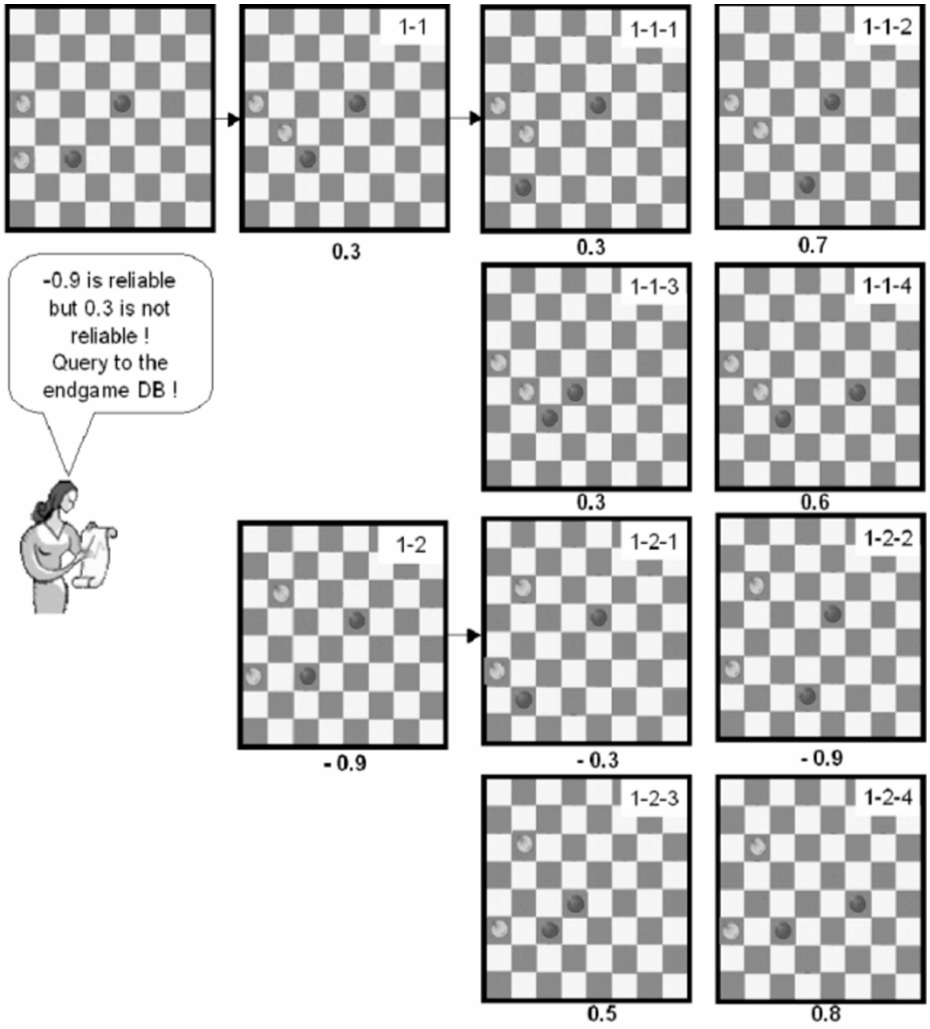


Fig. 14. Example of game tree with selective use of endgame database (two ply). The real value below the board represents the evaluation value of the neural network. Min operation is used to select the value of the board configuration at the level 1. If the evaluated value of the root node of a game tree using min–max algorithm is near 1 or -1, there is no need of endgame DB but in the case of vagueness (0.3), the use of endgame DB is needed.

the competition results are used for the selection). Evolving checkers using speciation requires 10 hours on a Pentium III 800 MHz (256 MB RAM). The nonspeciated EA uses only mutation but the speciated EA uses crossover and mutation. The nonspeciated EA is the same as Chellapilla and Fogel's checkers program. Table II summarizes parameters of a simple EA and a speciated EA. They have the same number of individuals for evolution and the game that they played for one generation is the same to ours.

TABLE II
PARAMETERS OF EXPERIMENTS

	Simple EA	Speciated EA
Population Size	100	100
Mutation rate	1.0	0.01
Crossover rate	-	1.0
Generation	50	50

Fig. 15 shows experimental methods when the number of clusters is fixed. Table III shows the results of experiments when the number of clusters is fixed. It is the result of 68 runs. In this result, the best single player of simple EA is a bit better than the best single player of speciated EA but it is not statistically significant. After combining speciated players, the performance gap between the best single player of simple EA and the coalition of players with speciated EA is large. The result is statistically significant.

Fig. 16 shows experimental methods when the number of clusters is not fixed. The DBSCAN algorithm is used for clustering and it can automatically select the number of clusters based on the predefined parameters. Table IV shows that the coalition of players with speciated EA produces better performance than the best single player with simple EA and speciated EA. Also, the coalition of players with speciated EA performs better than the coalition of players with simple EA. Each of the results is statistically significant.

Fig. 17 shows the average number of clusters for speciated and simple evolution. Fig. 18 shows a dendrogram of the population evolved using the speciation method. A dendrogram is used to understand the diversity of the population. Drawing a dendrogram requires computing the dissimilarity between

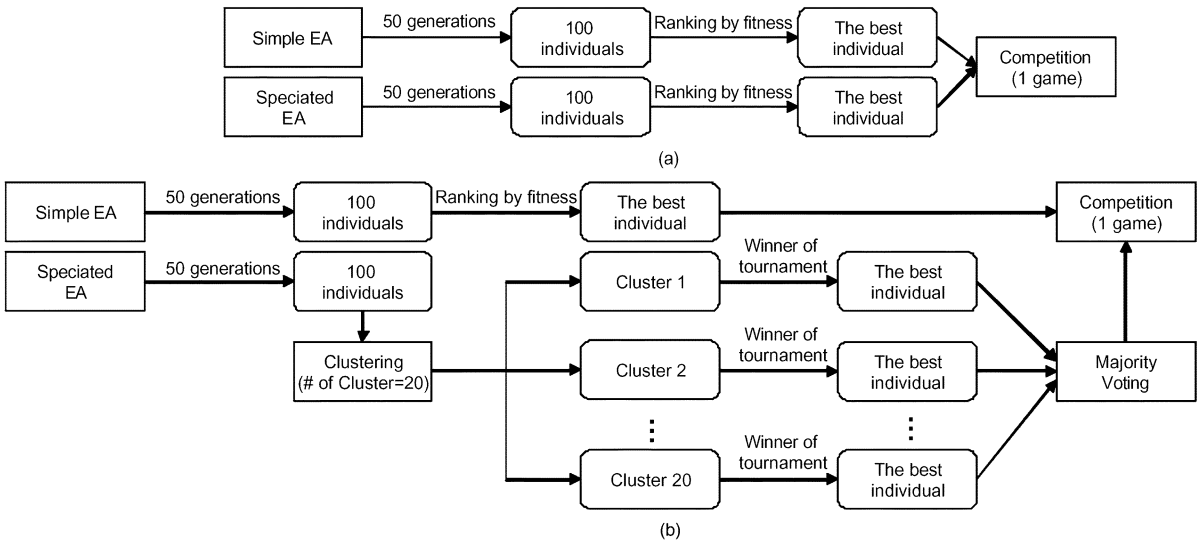


Fig. 15. Experimental design with the fixed number of clusters. (a) The best single player with simple EA versus the best single player with speciated EA. (b) The best single player (simple EA) versus the coalition of 20 players (speciated EA).

TABLE III
RESULTS OF EXPERIMENTS WHEN THE NUMBER OF CLUSTERS IS FIXED (68 RUNS). THE OPPONENT IS THE BEST SINGLE PLAYER IN THE LAST GENERATION THE SIMPLE EA. Z-SCORE INDICATES THAT THE z-STATISTIC FOR A PROBABILITY TEST (NULL HYPOTHESIS OF $p_0 = 0.5$) AND THE RESULTS ARE STATISTICALLY SIGNIFICANT ($\alpha = 0.05, z^* = +1.65$; NOTE THAT A ONE-SIDED TEST IS REPORTED HERE). WIN% INDICATES THE RATIO OF WINS

	Win	Lose	Draw	Win%	Z-score	Statistical Significance
A single best player (Speciated EA)	22	24	22	47.82	-0.29	X
The coalition of 20 players (Speciated EA)	37	16	15	69.81	2.88	O

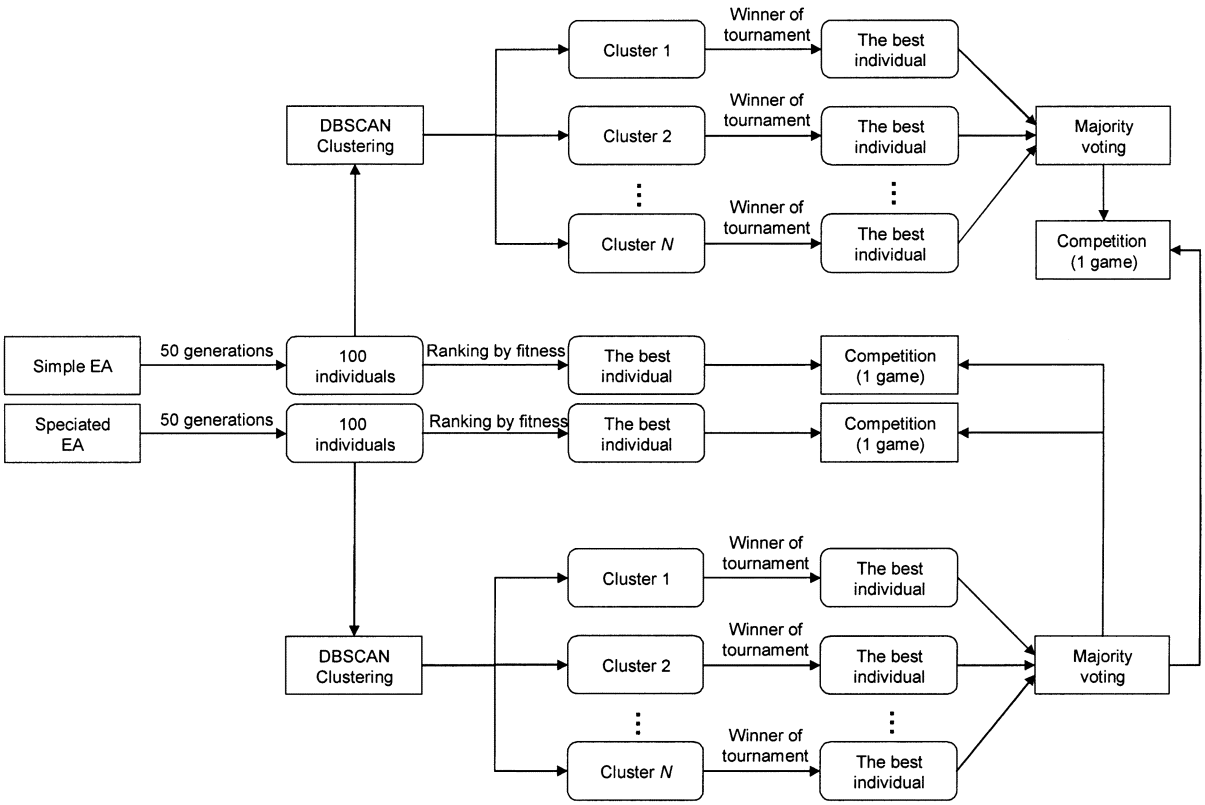


Fig. 16. Experimental design without the fixed number of clusters.

TABLE IV
SUMMARY OF EXPERIMENTAL RESULTS IN VARIOUS CONFIGURATIONS (250 RUNS). IN THIS EXPERIMENT, THE SIZE OF THE COALITION IS NOT FIXED

Player	Opponent	Win	Lose	Draw	Win%	Z-score	Statistical Significance
The coalition of players (Speciated EA)	The best single player (Simple EA)	98	76	76	56.32	1.66	O
The coalition of players (Speciated EA)	The best single player (Speciated EA)	108	73	69	59.66	2.60	O
The coalition of players (Speciated EA)	The coalition of players (Simple EA)	111	22	117	83.45	7.71	O

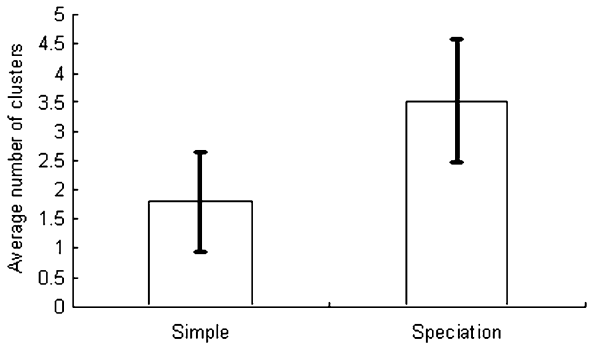


Fig. 17. Average number of clusters in simple and speciated evolution for 15 runs.

two objects in a population and performing single-linkage clustering. The behavioral characteristics of each individual are used as the measure of the dissimilarity. Though the method requires more computational time than is need to compute the Euclidean distance of two chromosomes, it is more accurate in this problem. Each individual is represented as a vector of 100 elements and the ith element represents the result of a game between the individual and the ith one. The Euclidean distance between two vectors is used to calculate the dissimilarity.

The Chinook endgame DB (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 [Fig. 19(a)] and the knowledge-based evolution takes a little more time than that without knowledge in the speciated evolution [Fig. 19(b)]. This means that the insertion of knowledge within a limited scope can accelerate the speed of the EA 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 V summarizes the competition results between the best individual in the evolution with knowledge and the best individual in the evolution without knowledge for each generation. The knowledge incorporation model performs better than the one without knowledge. Table VI shows the competition results in the speciated evolution. Table VII shows the effect of the stored knowledge (opening and endgame DB) in speciation.

TABLE V
EXPERIMENTAL RESULTS ON OPENING AND ENDGAME KNOWLEDGE INCORPORATION (WIN/LOSE/DRAW) FOR SIMPLE EVOLUTION. EVOLUTION WITH THE STORED KNOWLEDGE PERFORMS BETTER THAN THAT WITHOUT THE KNOWLEDGE

Simple EA with opening and endgame DB	Simple EA	Generations				
		1~15	16~30	31~45	46~60	Total
Red	White	5/0/10	3/3/9	3/0/12	5/3/7	16/6/38
White	Red	4/3/8	4/2/9	5/4/6	4/2/9	17/11/32

TABLE VI
EXPERIMENTAL RESULTS ON OPENING AND ENDGAME KNOWLEDGE INCORPORATION (WIN/LOSE/DRAW) FOR SPECIATED EVOLUTION. EVOLUTION WITH THE STORED KNOWLEDGE PERFORMS BETTER THAN THAT WITHOUT THE KNOWLEDGE

Speciated EA with opening and endgame DB	Speciated EA	Generations				
		1~15	16~30	31~45	46~60	Total
Red	White	5/1/9	4/3/8	6/0/9	8/2/5	23/6/31
White	Red	7/3/5	5/2/8	8/4/3	6/2/7	26/11/23

TABLE VII
COMPETITION RESULTS BETWEEN THE SPECIATED PLAYERS USING BOTH OPENING AND ENDGAME DB AND THE SPECIATED PLAYER WITH ONE OF THE KNOWLEDGE

Speciated EA with opening and endgame DB	Speciated EA with opening DB	Total
Red	White	6/2/7
White	Red	8/4/3
Speciated EA with opening and endgame DB	Speciated EA with endgame DB	Total
Red	White	5/5/5
White	Red	4/5/6
Speciated EA with opening DB	Speciated EA with endgame DB	Total
Red	White	3/6/6
White	Red	2/7/6

V. CONCLUSION

In this paper, evolved neural networks are used to evaluate configurations of a checkerboard. Like other problems, evolving checkers strategies benefit from diversity in a population. To improve the diversity of the population, a crowding algorithm is applied to the original EA. In the last generation, we cluster the individuals of population and choose one representative player from each species. From the experimental result, players

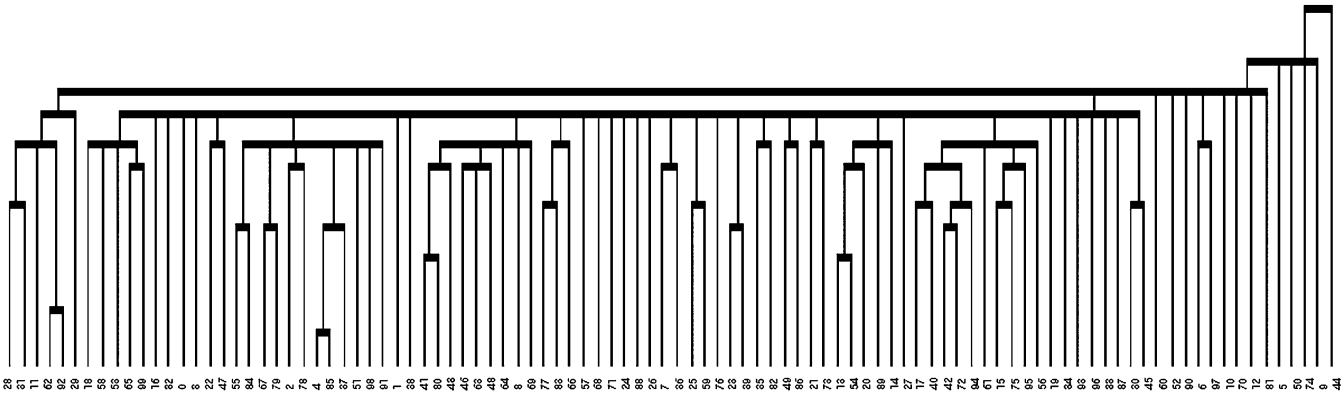


Fig. 18. Dendrogram of speciated population.

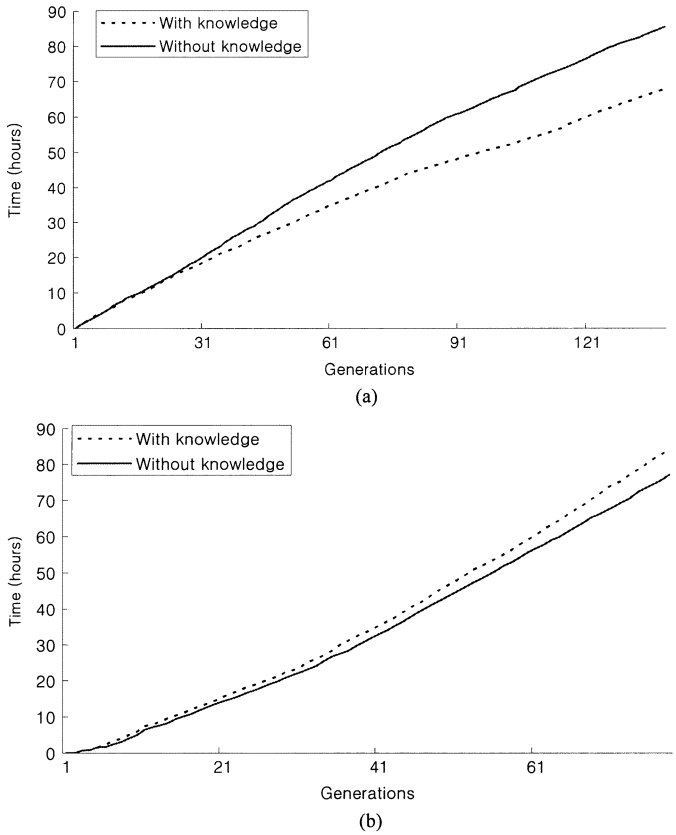


Fig. 19. Comparison of running time. (a) Simple evolution. (b) Speciated evolution.

evolved using the speciation method show higher performance than the best single player with simple EA. Additionally, the incorporation of domain-specific knowledge into the evolutionary procedure improves its speed and performance.

The final conclusion of the experiment is $\text{Simple EA} < \text{Speciated EA} < \text{Speciated EA with opening DB} < \text{Speciated EA with endgame DB} \approx \text{Speciated EA with opening and endgame DB}$. The effect of opening knowledge is not so significant because these have only limited sequences. The limited opening knowledge can prevent a player from making a big mistake but becomes useless when the opponent chooses a move that is not included in the opening sequence. For better performance, extensive opening knowledge must be used. Though previous evolutionary checkers reported two wins, four losses, and four draws against Chinook [25] it was evolved for 840 generations (six months with a Pentium II 400 Hz machine). Preparing for the game between Chinook (novice version) and speciated EA with opening and endgame DB requires the additional efforts such as increasing the number of games for fitness evaluation, optimizing game tree search, applying a game tree extension (two-ply) technique in [22], and evolving for much longer time. The focal point of this paper is to investigated the effect of the systematical incorporation of domain knowledge into evolutionary checkers, and we compared the variants of versions with or without knowledge and with or without speciation. Though it fails to achieve the level of Chinook, it gives an insight into improving basic evolutionary checkers in a unified framework.

Multiple diverse neural networks can perform better than the single best one, but there is always the problem of combination and averaging may not work. As future work, a sophisticated combination method should be explored for better performance.

ACKNOWLEDGMENT

The authors would like to thank the three anonymous reviewers and Dr. D. Fogel for their helpful and constructive comments.

REFERENCES

[1] G. Tesauro, "Programming backgammon using self-teaching neural nets," *Artif. Intell.*, vol. 134, no. 1–2, pp. 181–199, Jan. 2002.
[2] M. Campbell, A. J. Hoane Jr., and F.-H. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, no. 1–2, pp. 57–83, 2002.
[3] J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers*. New York: Springer-Verlag, 1997.
[4] M. Buro, "The othello match of the year: Takeshi Murakami vs. Logistello," *Int. Comput. Game Assoc. J.*, vol. 20, no. 3, pp. 189–193, 1997.
[5] D. B. Fogel, *Blondie24: Playing at the Edge of AI*. San Mateo, CA: Morgan Kaufmann, 2001.
[6] S. Y. Chong, D. C. Ku, H. S. Lim, M. K. Tan, and J. D. White, "Evolved neural networks learning Othello strategies," in *Proc. Congr. Evol. Comput.*, vol. 3, 2003, pp. 2222–2229.
[7] C.-T. Sun and M.-D. Wu, "Multi-stage genetic algorithm learning in game playing," *NAFIPS/IFIS/NASA*, pp. 223–227, 1994.
[8] G. Kendall and C. Smith, "The evolution of blackjack strategies," in *Proc. Congr. Evol. Comput.*, vol. 4, 2003, pp. 2474–2481.
[9] N. Richards, D. Moriarty, and R. Miikkulainen, "Evolving neural networks to play go," *Appl. Intell.*, vol. 8, pp. 85–96, 1998.

[10] G. Kendall and G. Whitwell, "An evolutionary approach for the tuning of a chess evaluation function using population dynamics," in *Proc. Congr. Evol. Comput.*, vol. 2, 2001, pp. 995–1002.
[11] J. B. Pollack and A. D. Blair, "Co-evolution in the successful learning of backgammon strategy," *Mach. Learn.*, vol. 32, no. 3, pp. 225–240, 1998.
[12] Y. Jin, *Knowledge Incorporation in Evolutionary Computation*. New York: Springer-Verlag, 2004.
[13] K.-J. Kim and S.-B. Cho, "Evolving speciated checkers players with crowding algorithm," in *Proc. Congr. Evol. Comput.*, vol. 1, 2002, pp. 407–412.
[14] K.-J. Kim and S.-B. Cho, "Checkers strategy evolution with speciated neural networks," in *Proc. 7th Pacific Rim Int. Conf. Artif. Intell.*, 2002, p. 599.
[15] I. Chernev, *The Compleat Draughts Player*. London, U.K.: Oxford Univ. Press, 1981.
[16] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron, "A world championship caliber checkers program," *Artif. Intell.*, vol. 53, no. 2–3, pp. 273–290, 1992.
[17] J. Schaeffer, R. Lake, P. Lu, and M. Bryant, "Chinook: The man-machine world checkers champion," *AI Mag.*, vol. 17, no. 1, pp. 21–29, 1996.
[18] D. B. Fogel, T. J. Hays, S. Hahn, and J. Quon, "A self-learning evolutionary chess program," in *Proc. IEEE*, vol. 92, 2004, pp. 1947–1954.
[19] R. Lake, J. Schaeffer, and P. Lu, "Solving large retrograde analysis problems using a network of workstations," in *Proc. Adv. Computer Chess VII*, 1994, pp. 135–162.
[20] D. B. Fogel, "Evolutionary entertainment with intelligent agents," *IEEE Comput.*, vol. 36, no. 6, pp. 106–108, Jun. 2003.
[21] K. Chellapilla and D. B. Fogel, "Evolving neural networks to play checkers without relying on expert knowledge," *IEEE Trans. Neural Netw.*, vol. 10, no. 6, pp. 1382–1391, Nov. 1999.
[22] ——, "Evolving an expert checkers playing program without using human expertise," *IEEE Trans. Evol. Comput.*, vol. 5, no. 4, pp. 422–428, Aug. 2001.
[23] D. B. Fogel, "Evolving a checkers player without relying on human experience," *ACM Intell.*, vol. 11, no. 2, pp. 20–27, 2000.
[24] K. Chellapilla and D. B. Fogel, "Anaconda defeats hoyle 6-0: A case study competing an evolved checkers program against commercially available software," in *Proc. Congr. Evol. Comput.*, vol. 2, 2000, pp. 857–863.
[25] D. B. Fogel and K. Chellapilla, "Verifying Anaconda's expert rating by competing against Chinook: Experiments in co-evolving a neural checkers player," *Neurocomputing*, vol. 42, no. 1–4, pp. 69–86, 2002.
[26] D. E. Moriarty and R. Miikkulainen, "Discovering complex Othello strategies through evolutionary neural networks," *Connection Sci.*, vol. 7, pp. 195–209, 1995.
[27] ——, "Evolving neural networks to focus minimax search," in *Proc. of the 12th National Conf. on Artif. Intell. (AAAI-94)*, 1994, pp. 1371–1377.
[28] ——, "Improving game-tree search with evolutionary neural networks," in *Proc. 1st IEEE Conf. Evolu. Comput.*, vol. 1, 1994, pp. 496–501.
[29] K. O. Stanley and R. Miikkulainen, "Evolving a roving eye for go," in *Proc. Genetic Evol. Comput. Conf.*, 2004, pp. 1226–1238.
[30] A. Lubberts and R. Miikkulainen, "Co-evolving a go-playing neural networks," in *Proc. Genetic Evol. Comput. Conf. Workshop Prog.*, 2001, pp. 14–19.
[31] A. S. Perez-Bergquist, "Applying ESP and Region Specialists to Neuro-Evolution for Go," Dept. Comput. Sci., Univ. Texas at Austin, Austin, TX, May 2001. Tech. Rep. CSTR01-24.
[32] L. Barone and L. While, "An adaptive learning model for simplified poker using evolutionary algorithms," in *Proc. Congr. Evol. Comput.*, vol. 1, 1999, pp. 153–160.
[33] D. B. Fogel, "Evolving strategies in blackjack," in *Proc. Congr. Evol. Comput.*, 2004, pp. 1427–1434.
[34] W. Ono and Y.-J. Lim, "An investigation on piece differential information in co-evolution on games using Kalah," in *Proc. Congr. Evol. Comput.*, vol. 3, 2003, pp. 1632–1638.
[35] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 10, pp. 993–1001, Oct. 1990.
[36] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron, "Reviving the game of checkers," *Second Comput. Olympiad on Heuristic Program. Artif. Intell.*, pp. 119–136, 1991.
[37] X. Yao and Y. Liu, "Evolving neural network ensembles by minimizing of mutual information," *Int. J. Hybrid Intell. Syst.*, vol. 1, no. 1, pp. 12–21, Jan. 2004.
[38] *Handbook of Evolutionary Computation*, Oxford Univ. Press, London, U.K., 1997. C6.1 S. W. Mahfoud Niching methods.
[39] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," *Knowl. Discovery and Data Mining*, pp. 226–231, 1996.

Kyung-Joong Kim (S'02) received the B.S. and M.S. degrees in computer science from Yonsei University, Seoul, Korea, in 2000 and 2002, respectively. He is currently working towards the Ph.D. degree in the Department of Computer Science, Yonsei University.

His research interests include evolutionary neural network, robot control, and agent architecture.

Sung-Bae Cho (S'88–M'98) received the B.S. degree in computer science from Yonsei University, Seoul, Korea, in 1988 and the M.S. and Ph.D. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST), Taejeon, Korea, in 1990 and 1993, respectively.

From 1991 to 1993, he worked as a Member of the Research Staff at the Center for Artificial Intelligence Research, KAIST. From 1993 to 1995, he was an Invited Researcher of Human Information Processing Research Laboratories, Advanced Telecommunications Research (ATR) Institute, Kyoto, Japan. In 1998, he was a Visiting Scholar at the University of New South Wales, Canberra, Australia. Since 1995, he has been a Professor in the Department of Computer Science, Yonsei University. His research interests include neural networks, pattern recognition, intelligent man-machine interfaces, evolutionary computation, and artificial life.

Dr. Cho is a Member of the Korea Information Science Society, INNS, the IEEE Computer Society, and the IEEE Systems, Man and Cybernetics Society. He was awarded outstanding paper prizes from the IEEE Korea Section in 1989 and 1992, and another one from the Korea Information Science Society in 1990. In 1993, he also received the Richard E. Merwin Prize from the IEEE Computer Society. In 1994, he was listed in *Who's Who in Pattern Recognition* from the International Association for Pattern Recognition and received the Best Paper Awards at the International Conference on Soft Computing in 1996 and 1998. In 1998, he received the Best Paper Award at the World Automation Congress. He was listed in *Marquis Who's Who in Science and Engineering* in 2000 and in *Marquis Who's Who in the World* in 2001.