An evolutionary learning approach to play Othello using XCS

Satvik Jain*, Siddharth Verma*, Swaraj Kumar* and Swati Aggarwal†
Department of Computer Engineering
Netaji Subhas Institute of Technology, Delhi, India
siddharth.verma60@gmail.com, jainsatvik97@gmail.com, swaraj01kumar@gmail.com, swati1178@gmail.com
*Equal contribution †Corresponding author

***Abstract*—Due to the multifarious challenges that emerge when developing an artificial intelligent (AI) agent that can compete with human players, the classic game of Othello has received a lot of attention from the Computational-Intelligence community. This paper proposes an AI agent that learns a winning strategy for the game of Othello using the eXtended Classifier System (XCS) algorithm which is a popular variant of the Learning Classifier System (LCS) algorithm. Othello has been a favourite in the study of AI due to its simple set of rules, low branching factor and well defined strategic concepts. The LCS system consists of a rule-set which is made to evolve using a combination of Reinforcement Learning (RL) and Genetic Algorithm (GA) such that the evolved rule-set learns an optimal action for each input board state. A 6×6 Othello board will be used for this experiment in order to evaluate the applicability of the proposed agent in learning a winning game-playing strategy. The performance of the proposed agent was evaluated against three categories of agents: minimax, human and random agent. The XCS agent was able to outperform the above-mentioned agents showing the effectiveness of rule-based evolutionary learning in Othello. This work demonstrates the possibility of using the XCS algorithm in other strategy-based combinatorial games.**

***Keywords*—eXtended Classifier System, Learning Classifier System, Othello, Computational Intelligence, Evolutionary Computation.**

I. Introduction

Othello is a combinatorial board game played on an 8×8 uncheckered board. Combinatorial games involve two players having sequential moves competing with perfect information about each game state. Combinatorial games like Chess and Go have long been the subject of rigorous AI research. AI techniques such as deep reinforcement learning and sophisticated search techniques [1], [2] are generally applied in such games to model agents that could perform at the level of their human competitors. One extensively researched technique is LCS that uses a rule-based mechanism to explore the state space and evolve its rule-set to come up with the optimal action for each input state.

LCS was developed as the biologically inspired computational model for human-like cognition. It is one of the classical computational intelligence approaches [3]. John Holland introduced the concept of classifier systems having adaptive learning capabilities [4]–[6]. He developed the system which can build model representation of the environment, improve itself from experience, and make itself better through breeding and generalization. He used a rule-based representation of the environment. GA [4] was used for adaptive evolution of new rules and RL was used to test the effectiveness of existing rules. Rules are typically represented in the form of "IF condition THEN action". This makes the classifier systems a highly interpretable model which adequately describes the environment with the rule-set. Holland summarized [5] classifier systems as rule based systems having mechanisms to process rules in parallel, adaptively generate new rules and test the efficiency of existing rules.

Recent studies and surveys [7], bring forth the application of LCS in games [8], [9]. Classifier systems form an easy integration with the rule-based knowledge used in games. Also, recent advances in LCS have made it more capable of performing in real time complex environments with reasonable efficacy [10], [11]. In the past, LCS algorithm has not been applied much to combinatorial games [7]. One of the significant works involving the use of the LCS algorithm in a board game is by Browne and Scott [12]. They explored the performance of XCS algorithm in the game of Connect4. XCS algorithm was introduced by Wilson in 1995 [13] which was a modification of LCS and used accuracy-based fitness with GA in action-set for improved performance.

Various AI techniques having different combinations of heuristic functions and optimization techniques have been used in the past to create Othello playing agents. Game-search technique such as minimax [14], RL techniques [15], [16] such as Q-learning, Sarsa and TD-learning, and neural networks [17] have been used to learn strategies for playing Othello. The aim of this paper has been to create an evolutionary learning agent based on the XCS framework which uses rules that evolve over time to learn a winning game-playing strategy for Othello. In this work, the 8×8 board has been reduced to the size of 6×6 for evaluation of the proposed agent. This has been done to reduce the number of states and possible actions in the game, which expediates time required to train the agent.

II. Background

In this section, the game of Othello and it's different game-playing strategies are explained. Further, a description of the XCS algorithm is given along with a brief summary of its working cycle.

A. Othello

Othello is a perfect-information strategy based game which is played on a 64-square board among two competing players having black and white discs respectively. Objective of each player is to maximize their discs on the board by end of the game. Winner is the player who ends the game with highest number of own discs. Each player takes its respective discs (Black/White) and begins the game with the setup shown in Fig. 1. Here pink cells show the valid moves of current player (i.e. Black player in Fig. 1) that can be taken in that particular input board state.

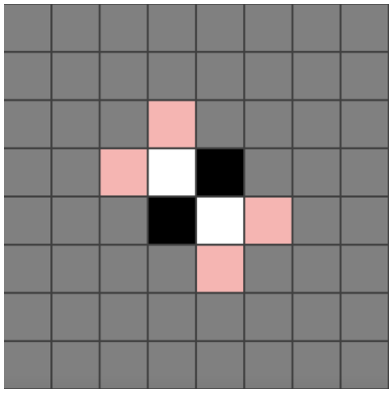

Fig. 1. Initial state of the Othello board

A move consists of outflanking the opponent's disc(s) and then flipping the outflanked disc(s) to the player's color (Fig. 2). Outflanking means to place a disc on the board so that the opponent's row/column/diagonal of disc(s) is bordered at each end by the disc of the player.

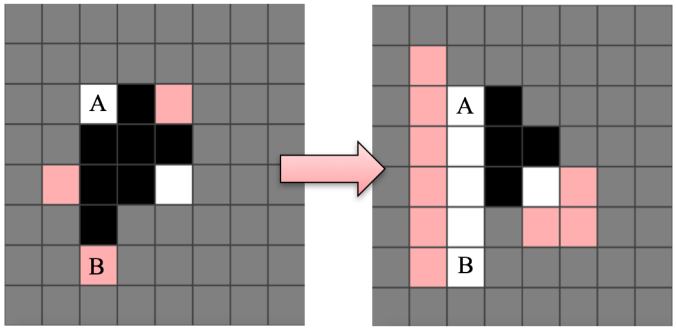

Fig. 2. Illustrating the process of black discs getting outflanked and consequently sandwiched between A and B during whites turn. As soon as white plays on position B, the corresponding column is flipped to white.

The greedy strategy commonly adopted for playing Othello is to capture the maximum number of opponent-discs in each move. However, this is not advisable, as towards the end, discs captured in initial moves are easily lost. Many game-playing strategies for Othello have been developed over time, the most popular among them being the positional strategy and the mobility strategy [15].

The positional strategy takes into account stability of discs at various positions on Othello board and accordingly assigns values to those positions as shown in Fig. 3. Position values assigned to the squares in 8×8 Othello board [15] have been symmetrically reduced to 6×6 in Fig. 3. This is possible because reduction in size does not change how games are played and also the position values in the original Othello board have rotational symmetry. The corner positions have highest value associated with them since discs placed at the corners cannot be flipped over in any move.

	A	B	C	D	E	F
1	50	-20	5	5	-20	50
2	-20	-50	-2	-2	-50	-20
3	5	-2	-1	-1	-2	5
4	5	-2	-1	-1	-2	5
5	-20	-50	-2	-2	-50	-20
6	50	-20	5	5	-20	50

Fig. 3. Position value of each cell in the Othello board.

Mobility in Othello is defined as the available board positions for a player in a given move. Mobility strategy is based on the premise of reducing opponent's mobility thus giving the opponent fewer options to capture high-value board positions.

B. Description of XCS

The most popular LCS algorithm is an RL based LCS named XCS [13]. RL and LCS share many common features. For instance, RL is used in sequential learning tasks where the agent learns through trial and error by performing an action and then receiving rewards as feedback from the environment. For evaluating rules in the LCS system, Holland, earlier used the Bucket Brigade algorithm, an archaic approach in RL literature [18]. With the advancements in RL such as the introduction of Temporal difference learning [19] and Q-learning [20], these newer algorithms were employed for assigning rewards to the rules. Like neural networks are used to approximate the value function in RL, the LCS rules also act as state-space approximators with an interpretable representation of the environment.

XCS is the variant of LCS which uses accuracy-based fitness value and Q-learning like update in multi-step problems. Classifiers form building blocks of the XCS framework. Each classifier in the population consists of three parts [21]: Condition, Action and Parameters. Fitness is one of the classifier parameters that reflects quality of the classifier. Earlier, fitness was strength based i.e. it was directly proportional to the total reward received from the environment. The introduction of accuracy based fitness was a major breakthrough in LCS literature which highly improved performance of the algorithm [13].

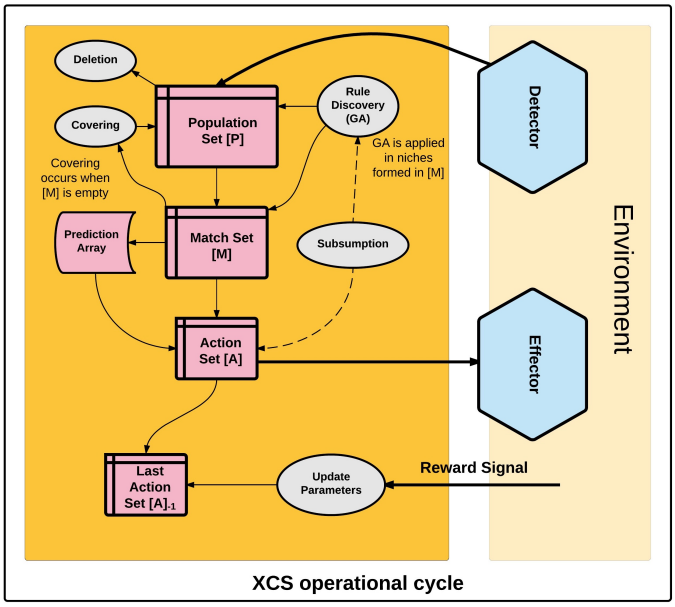

Fig. 4. Working steps of XCS cycle in a typical reinforcement learning environment. Pink boxes are the storage sets containing the classifiers, grey boxes represent processes of XCS cycle and the blue boxes are the environment components.

Fig. 4 depicts the working of the complete XCS cycle. An encoded input state is received from the environment and has same representation as that of the condition part of a rule. Most commonly used syntax for rule condition representation is fixed length bit-strings of the ternary alphabet (0, 1, #) [11]. # represents a don't care that in turn represents a general state, i.e. it can be 1 or 0. This input state is compared with the classifiers present in the population-set $[P]$. $[P]$ is initialized either randomly or by using the covering operator. Match-set $[M]$ is formed consisting of all the classifiers in $[P]$ whose condition matched the input state. This matching can be between each specified bit or some other criterion like distance can be used [22]. Now, if the match-set remains empty, then covering operator is called and the population is populated by creating multiple classifiers matching the current input and by introducing don't cares (#) with a specified probability. After this, action to be executed is selected from the classifiers in $[M]$. Action selection can be carried out by selecting any random action from $[M]$. This random selection of the action encourages exploration in the environment. Another way to choose an action is through exploitation which selects the best action so far. This is done using a prediction-array which is formed based on the prediction-value of the classifier as explained in Section III-C. Action with the highest prediction-value is chosen and action-set $[A]$ is formed comprising of classifiers advocating the selected action. The selected action is executed and a reward signal is given to all classifiers in the previous action-set $[A]_{-1}$ (action-set created in the previous time step). Reward assignment to the prediction-value of classifiers in $[A]_{-1}$ is done according to the following Q-learning update equation:

$$Q(s_{t-1}, a_{t-1}) \leftarrow (1-\alpha)Q(s_{t-1}, a_{t-1}) + \alpha(r_t + \gamma \max_{a \in A} Q(s_t, a)) \quad (1)$$

In XCS framework, $Q(s_{t-1}, a_{t-1})$ corresponds to the prediction-value of the classifiers in $[A]_{-1}$, r_t is the reward signal received from the environment at time-step t and $\max_{a \in A} Q(s_t, a)$ denotes maximum value in the prediction-array formed at time-step t. α and γ are the learning-rate and discount-factor respectively [23].

A GA cycle is applied in $[P]$ or $[M]$ to introduce plausibly better classifiers into the population which is done by applying genetic operations like mutation and crossover over the classifiers. Mutation seeks to change individual bit values of the classifier's condition. It diversifies the search space covered by either expanding or shrinking the search area. It either mutates specific bits to another specific bit (0 to 1 or 1 to 0) or makes the rule more general (0 or 1 to #). Conversely, the generality of search space can also be reduced (# to 1 or 0). Crossover seeks to incorporate the useful parts of the classifiers together. Good solutions searched by the XCS algorithm need to be found and joined together to make a highly effective classifier.Basic crossover considers the condition part of two classifiers (parents) and swaps bits between them to form a new classifier (offspring). Two-point crossover selects two crossover points to cut the two parents and swaps the middle section of the rule to form an offspring. Therefore, while mutation helps to variegate the search space, crossover helps to combine the effective knowledge together. Commonly, Roulette-wheel selection or Tournament selection is used to select the parents for breeding based on fitness values [24]. Applying GA in $[M]$ or $[A]$ is more intuitive as groups of similar classifiers are formed in $[M]$ or $[A]$ and evolving those classifiers by breeding them against each other will result in better generalized classifiers. Subsumption is a process by which a more general classifier, having a high fitness value, subsumes the more specific classifiers. Subsumption can either be applied directly in $[A]$ or with GA. Subsumption is however avoided in simpler problem domains. To allow space in the poulation for newly created classifiers, deletion through Tournament/Roulette-wheel selection is done by removing the less fit classifiers in $[P]$. This also ensures that size of the population does not increase to infinity.

III. Proposed Approach

A typical Othello board consists of 8×8 squares which results in 60 possible actions (64 squares with 4 initially occupied). In this work, a 6×6 subset of the original board has been used to establish the effectiveness of XCS algorithm for learning a winning game playing strategy in Othello. By doing so, the search space is reduced and maximum number of possible actions are restricted to 32. This shortens the time required to achieve desired results. Hence, a conceptual framework has been established for using XCS algorithm in Othello which can further be extended to the original board size.

The game-playing agent's framework and the algorithmic components used to design the agent have been described in this section.

A. Representation of the classifiers and the input state

Every classifier consists of the following components:

1) Condition: This is used to match the rule to the corresponding input state. Many different syntaxes have been examined for representing a rule condition such as fixed length bit-strings of the ternary alphabet (0, 1, #) [11], the real-valued alphabet [25] and fuzzy logic [26]. In the current model, the ternary alphabet string representation of rules has been used to encode the input state of Othello board. The following encoding technique is used to capture each state:

- 01 for white cell
- 10 for gray cell (Invalid move)
- 11 for pink cell (Valid move)
- # as Don't care symbol

2) Action: This part of the classifier specifies the action to be taken when input state matches with the condition of that classifier. For each classifier, action is defined in terms of exact coordinates of the cell on Othello board on which disk is to be placed in the current move.

3) Parameters: Parameters are used to determine the classifier's quality that indicates it's worth of being present in the population-set [21]. Their values evolve over time as the training progresses. Prediction-value, fitness, prediction-error, experience and numerosity are some of the important parameters that belong to every classifier in the population. Prediction-value gives the estimate of payoff expected if the classifier's action is executed. Fitness is a function of accuracy of the classifier and determines if classifier should be present in the population. The prediction-error gives the value of errors made in the prediction and is calculated using the difference between input reward and expected payoff of the classifier. Experience counts the number of times a classifier has been a part of the action-set since its formation and numerosity refers to the number of micro-classifiers (more specific classifiers) that a macro-classifier (more general classifier) represents.

B. Learning and Evolutionary components

RL (the learning component) and GA (the evolutionary component) are the two building-blocks of XCS algorithm. The contribution and significance of these two components in the evolution of rule population have already been illustrated in Section II-B. The following subsection explains how RL and GA have been used within the XCS framework to train the rule-based agent and learn the optimal game playing strategy for Othello.

1) Reinforcement Learning: Prediction-value of the classifiers in $[A]_{-1}$ are updated at each non-terminal time step through Q-learning update equation (1).

In order to define the reward r_t in (1), received at each intermediate time-step, game of Othello is divided into two phases according to the number of board positions filled at that particular time step:

a) Beginning and Middle phase: Less than 80% of the total positions/squares on Othello board have been filled.

During this phase, the reward r_t at an intermediate time step t is defined as:

$$r_t = \frac{w_1v_1 + w_2v_2 + w_3v_3 + ... + w_{36}v_{36}}{|v_1| + |v_2| + |v_3| + ... + |v_{36}|} \tag{2}$$

where w_i ($i \in [1, 36]$) is $+1$, -1 or 0 for each square depending on whether the board position is occupied by agent's disc, occupied by opponent's disc or unoccupied respectively. Value v_i for each square is given by the position value grid in Fig. 3.

b) End phase: More than 80% of the total positions/squares have been filled on the Othello board.

During this phase, the reward r_t at an intermediate time step t is defined as:

$$r_t = \frac{(n_a - n_o)}{36} \tag{3}$$

where n_a represents the number of discs belonging to the agent and n_o represents the number of discs belonging to opponent.

Both the above reward terms defined for a non-terminating time step, have been normalized such that $-1 < r_t < 1$. The values of the learning-rate α and the discount-factor γ in (1) have been chosen experimentally as 0.1 and 1.0 respectively. At the end of each game, if XCS agent wins the game, all classifiers in action sets formed during the game are given an end-reward of $+1$. If there is a tie, end-reward of 0 and in case of a loss, -1 end-reward is given to the classifiers (end-reward is the reward given once the game ends).

2) Genetic Algorithm: The condition for GA execution is checked at the end of each game on all action sets used during the game, given by [21]:

$$t - \frac{\sum_{C \in [A]} t_{sC} * num_C}{\sum_{C \in [A]} num_C} > \theta_{GA} \tag{4}$$

where t is the current game number, C is the classifier, num_C is numerosity of the classifier C, t_{sC} is time stamp and θ_{GA} is the GA threshold. Genetic operators are then applied on all action sets of each game which satisfy the above condition. Two-point cross-over is applied with crossover-probability 0.8 and mutation is applied with mutation-probability 0.4 in which parents are selected from the action-set based on fitness value through Roulette-wheel selection procedure. The off-springs generated are then checked for subsumption.

C. Algorithmic Description

Algorithm 1 shows step-wise explanation of complete XCS cycle in the game.

In the algorithm, $[P]$ denotes population-set, $[M]$ denotes match-set, $[PA]$ is the prediction-array containing prediction-value of the matched classifiers, $[A]$ represents action-set, $[A]_{-1}$ is the action-set formed in previous iteration and $[AC]$ is the collection of all $[A]$ formed during the game.

Algorithm 1 Running procedure of XCS in Othello

Require: $Othello$ board-state

$[P] \leftarrow []$, $[M] \leftarrow []$

$[M] \leftarrow$ GENERATEMATCHSET($[P]$)

if $\mu < \theta$ **OR** $[P]$ $is\ empty$ **then**
    Apply covering;
end if

$[PA] \leftarrow$ GENERATEPREDICTIONARRAY($[M]$);
$prediction \leftarrow$ EXPLOREEXPLOIT($[PA]$);
$action \leftarrow$ SELECTACTION($predcition$);
$[A] \leftarrow$ GENERATEACTIONSET($action$);

TAKEACTION($action$); ▷ The chosen action is executed on the Othello board

QREWARDUPDATE($[A]_{-1}, [PA]$);
UPDATEPARAMETERS($[A]_{-1}$); ▷ Fitness-value and prediction-error of classifiers in $[A]_{-1}$ are updated

SUBSUMPTION($[A], [P]$);
$[AC] \leftarrow$ ADDACTIONSET($[A]$);
if $Othelloiscomplete$ **then**
    $reward \in \{-1, 0, 1\}$;
    ENDREWARD($[AC], reward$);
    UPDATEPARAMETERS($[AC]$);
    APPLYGA($[AC]$);
end if

1. $[P]$ and $[M]$ are initially empty. $[P]$ is initialized using the covering mechanism when input state is received.
2. XCS agent receives a ternary-bit string representing the input state on Othello board.
3. Input state is compared with the condition of every classifier in $[P]$. If the classifier's condition matches with input state, it is included in $[M]$.
4. $[M]$ is then traversed and number of unique actions in the set is recorded as μ. If the value of $\mu < \theta$ (θ denotes number of valid actions available on the Othello board), covering operation is performed and $[P]$ is populated with the required number of classifiers.
5. After the formation of $[M]$, $[PA]$ is created containing all unique actions in $[M]$. For each action, a prediction value $P(a)$ is computed by taking the fitness-weighted average of all matching classifiers with that action [27].

$$P(a) = \frac{\sum_{C.a=a \wedge C \in [M]} C.p * C.F}{\sum_{C.a=a \wedge C \in [M]} C.F} \quad (5)$$

where $C.a, C.p$ and $C.F$ respectively represent action, prediction and fitness associated with the classifier C.

6. Action is chosen based on the $\epsilon - greedy$ policy of action selection which uses a combination of exploration (choosing the action randomly) and exploitation- (choosing the action with highest prediction value in prediction array). $[A]$ is formed using all classifiers in $[M]$ that proposes the selected action.
7. The selected action in $[A]$ is executed on the Othello board.
8. Prediction-value is updated for all classifiers present in $[A]_{-1}$ according to Q-learning equation(1) for which the reward terms have been defined in Section III-B.
9. After updating the prediction values, the prediction-error for all the classifiers is evaluated as follows [21]:

$$\epsilon_C \leftarrow \epsilon_C + \frac{(|r_t + \gamma \max[PA] - p_C| - \epsilon_C)}{exp_C} \quad (6)$$

where ϵ_C is the prediction-error, p_C is the prediction-value and exp_C is the experience of classifier.

10. Fitness-value of classifiers in $[A]$ is updated using the equation [21]:

$$f_C \leftarrow f_C + 0.2 * (\kappa_C * \frac{num_C}{accuracySum} - f_C) \quad (7)$$

where f_C is the fitness-value, κ_C is the accuracy (inversely proportional to the prediction-error), num_C is the numerosity and $accuracySum$ is the sum of accuracies of classifiers in $[A]$.

11. Action-set subsumption is then applied to the classifiers present in $[A]$ and $[P]$ is updated incorporating changes after subsumption.
12. Once the game is completed, all classifiers in [A] contained in [AC] are given the end-reward and their prediction errors and fitness values are updated. GA is also applied in [AC] as discussed in Section III-B.

IV. EVALUATION

XCS agent developed using the approach given in Section III was trained for 60,000 games against an opponent that played randomly. Population-set size was kept fixed at 50,000 and θ_{GA} was set to 50. Value of exploitation-probability(ϵ) was increased over the training period according to the sigmoid function such that the probability was 0 initially, 0.5 at 30,000 games and 1 at 60,000 games.

Fig. 5 shows the winning percentage of XCS agent against random agent during the training process. A continuous increase in the winning percentage of XCS agent is seen, however as the number of training games approach 60,000 the rate of increase is very minimal and winning percentage starts to saturate at 65%.

Fig. 6 shows variation in fitness-value of classifiers over the training period. Fitness-values of the classifiers in the population have been compared for two phases during the

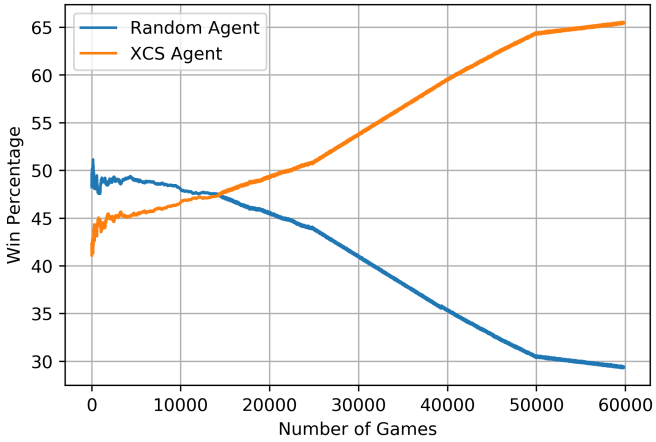

Fig. 5. Winning percentage of XCS agent against random agent during the training process.

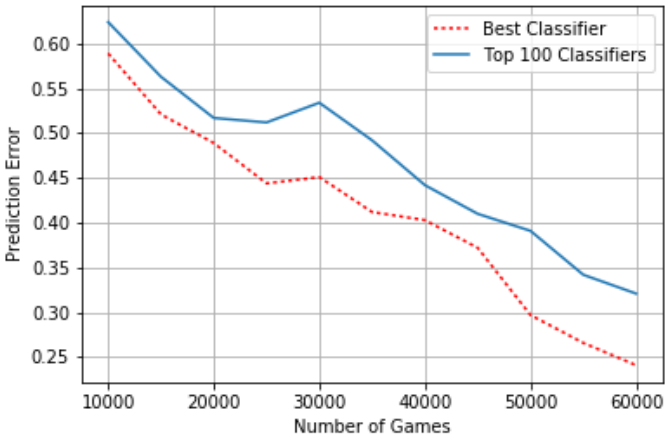

Fig. 7. Prediction-error of the best-fitness classifier and top 100 classifiers having highest fitness values.

training period: 1) Till 30,000 games and 2) 30,000-60,000 games. It can be seen that as the agent gets trained, mean fitness-value increases from approximately 0.0035 in the first 30,000 games to 0.0052 in the last 30,000 games. This is indicative of an improvement in quality of classifiers present in the population-set. Also, standard deviation in fitness values reduces for the second phase, indicating less variance in fitness of the classifiers from increased average.

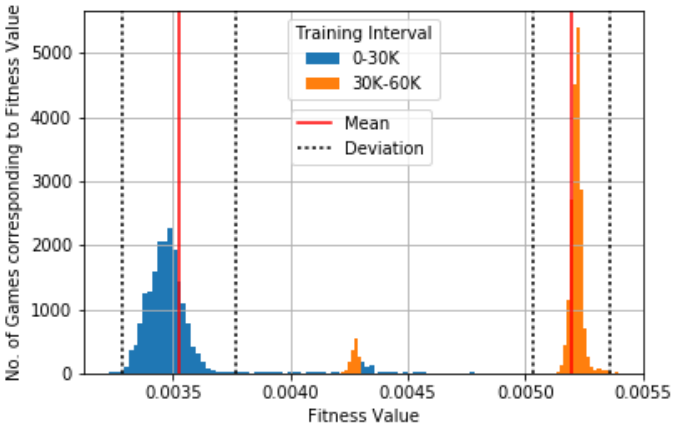

Fig. 6. Variation in fitness-value of classifiers over the training period.

Prediction-error of the best-fitness classifier and top 100 classifiers having highest fitness values have been shown in Fig. 7. A downward trend in the prediction-error of classifiers is observed as the number of games played increases.

Fig. 8 shows the number of wins of the XCS agent against the minimax agent operating at three different search-depths for a set of 100 games. The rule population evolved in the training process was used by the XCS agent against the minimax agent.

The proposed agent was also tested against human players. Since humans are the most unpredictable and non-deterministic agents, this served as a true test of the model's capability. Each human player was made to play 20 games

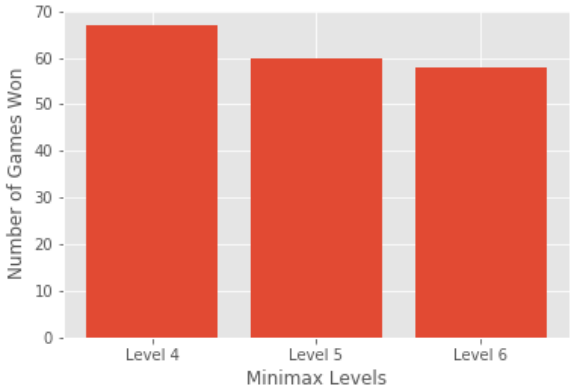

Fig. 8. Number of wins of the XCS agent operating at three different search-depths of the minimax agent.

each. Nine humans with varying skill levels competed against the XCS agent. Three of them were experienced Othello players, three players had average skills and played Othello for recreational purposes and the rest were new to Othello and had low strategic knowledge of the game. Each participant was also made to play an additional five games before beginning the test to get them accustomed to the game GUI. Table I depicts the performance of XCS agent against humans.

TABLE I
PERFORMANCE OF XCS AGENT AGAINST HUMANS.

Skill level	Wins	Loss	Ties
Highly skilled	40	11	9
Average skilled	46	5	9
Low skilled	55	2	3

V. CONCLUSION

Through this work, the effectiveness of using XCS algorithm to construct a game-playing AI agent for Othello has been established by obtaining favorable results when made

to compete against 3 different categories of agents: minimax, human and random agent. A new approach of applying Reinforcement Learning and Genetic Algorithm is introduced to effectively use XCS algorithm in the large state-space offered by Othello board. This approach ensures that all the classifiers which contribute to a winning game are rewarded and high-accuracy classifiers are successively introduced into the population set, throughout the training process. The fact that the agent was able to beat other Othello playing agents by learning a winning strategy is a testament to the proposed combination of RL and GA as an evolutionary learning mechanism in a rule-based format. This work is essentially a proof-of-concept that the XCS algorithm can be applied to play Othello, and an exploratory foray into how to actually do so. Also, this successful implementation of the XCS framework in 6×6 Othello serves as a reference for possible future use of this algorithm in other classic combinatorial games.

VI. Future Directions

Future work would involve expanding the search space to address the full 8×8 Othello game board instead of the reduced 6×6 version. Since the 8×8 board of Othello has a large action space, therefore introducing generalization over the action space and replacing the prediction-value parameter with a parametrized function [28] could significantly improve the agent's performance and provide better results. Also, training could be carried with other state of the art agents involving AI techniques such as neural networks and RL and more comparative results can be computed and analyzed.

Further, the rule-based or symbolic nature of the XCS algorithm renders it the unique ability to reason in games. The condition part of a rule gives information about the input state which can be used to draw the reason behind taking the corresponding action for that state. This level of reasoning can be used to explain actions for users which can further be used to diagnose their performance and conduct post-action reviews. This possibility of using LCS (or some variation of it) for "user diagnosis" can prove to be especially useful in enhancing cognitive control through the use of video games and simulation games.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[3] L. Bull, "Learning classifier systems: A brief introduction," *Applications of Learning Classifier Systems*, pp. 1–12, 2004.

[4] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[5] H. J. H. K. N. RE and P. Thagard, "Induction: Processes of inference, learning, and discovery cambridge," 1986.

[6] J. Holland, "Adaptation, w: Progress in theoretical biology, red. r. rosen, fm snell," 1976.

[7] K. Shafi and H. A. Abbass, "A survey of learning classifier systems in games," *IEEE Computational Intelligence Magazine*, vol. 12, no. 1, pp. 42–55, 2017.

[8] S. Rudolph, S. von Mammen, J. Jungbluth, and J. Hähner, "Design and evaluation of an extended learning classifier-based starcraft micro ai," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2016, pp. 669–681.

[9] C. Castillo, M. Lurgi, and I. Martinez, "Chimps: An evolutionary reinforcement learning approach for soccer agents," in *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, vol. 1. IEEE, 2003, pp. 60–65.

[10] O. Sigaud and S. W. Wilson, "Learning classifier systems: a survey," *Soft Computing*, vol. 11, no. 11, pp. 1065–1078, 2007.

[11] R. J. Urbanowicz and J. H. Moore, "Learning classifier systems: a complete introduction, review, and roadmap," *Journal of Artificial Evolution and Applications*, vol. 2009, p. 1, 2009.

[12] W. Browne and D. Scott, "An abstraction agorithm for genetics-based reinforcement learning," in *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, 2005, pp. 1875–1882.

[13] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary computation*, vol. 3, no. 2, pp. 149–175, 1995.

[14] C. N. Silla, M. Paglioney, and I. G. Mardegany, "jothellot: A java-based open source othello framework for artificial intelligence undergraduate classes," in *Frontiers in Education Conference (FIE), 2016 IEEE*. IEEE, 2016, pp. 1–7.

[15] N. J. van Eck and M. van Wezel, "Reinforcement learning and its application to othello," *Department of Computer Science, Faculty of Economics, Erasmus University, The Netherlands*, 2004.

[16] M. Van Der Ree and M. Wiering, "Reinforcement learning in the game of othello: learning against a fixed opponent and learning from self-play," in *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*. IEEE, 2013, pp. 108–115.

[17] V. Makris and D. Kalles, "Evolving multi-layer neural networks for othello," in *Proceedings of the 9th Hellenic Conference on Artificial Intelligence*. ACM, 2016, p. 26.

[18] J. Holland and J. Reitman, "Cognitive systems based on adaptive algorithms in pattern-directed inference systems. waterman and hayes-roth editors," 1978.

[19] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[20] C. Watkins, "C. h., and p. dayan,q-learning," *Machine Learning*, vol. 8, pp. 257–277, 1992.

[21] M. V. Butz and S. W. Wilson, "An algorithmic description of xcs," in *International Workshop on Learning Classifier Systems*. Springer, 2000, pp. 253–272.

[22] K. Shafi, H. A. Abbass, and W. Zhu, "The role of early stopping and population size in xcs for intrusion detection," *Lecture notes in computer science*, vol. 4247, p. 50, 2006.

[23] P. L. Lanzi, "Learning classifier systems from a reinforcement learning perspective," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 6, no. 3, pp. 162–170, 2002.

[24] J. Zhong, X. Hu, J. Zhang, and M. Gu, "Comparison of performance between different selection strategies on simple genetic algorithms," in *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, vol. 2. IEEE, 2005, pp. 1115–1121.

[25] L. Cielecki and O. Unold, "Gcs with real-valued input," *Bio-inspired Modeling of Cognitive Tasks*, pp. 488–497, 2007.

[26] A. Bonarini, "An introduction to learning fuzzy classifier systems," in *International Workshop on Learning Classifier Systems*. Springer, 1999, pp. 83–104.

[27] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "Toward a theory of generalization and learning in xcs," *IEEE transactions on evolutionary computation*, vol. 8, no. 1, pp. 28–46, 2004.

[28] P. L. Lanzi and D. Loiacono, "Classifier systems that compute action mappings," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 1822–1829.