Deck Building in Collectible Card Games using Genetic Algorithms: A Case Study of Legends of Code and Magic

Ya-Ju Yang
Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, Taiwan, R.O.C.
yangyalu.w@gmail.com

Tsung-Su Yeh
Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, Taiwan, R.O.C.
yehtsungsu@gmail.com

Tsung-Che Chiang
Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, Taiwan, R.O.C.
tcchiang@ieee.org

2021 IEEE Symposium Series on Computational Intelligence (SSCI) | 978-1-7281-9048-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/SSCI50451.2021.9659984

***Abstract*—Collectible card games (CCGs) are a kind of game in which players use a variety of cards to achieve the goal of game, for example, defeating the opponent. To play a game, players need to build a card deck, which consists of specified number of cards; only cards in the deck can be used in the game. Deck building is an important part in playing CCGs. This paper addresses deck building in a recently released CCG, Legends of Code and Magic. Our strategy is to evaluate cards by some scoring functions and then select cards into the deck accordingly. We adopt three types of scoring functions and use genetic algorithms (GA) to set proper values of parameters in the scoring functions. We checked the performance of the three versions of deck-building GAs against existing agents and found that scoring by card attributes is effective. We also analyzed the decks built by these algorithms and discussed their pros and cons.**

***Keywords*—genetic algorithm, collectible card game, deck building, Legends of Code and Magic**

I. Introduction

Games are full of fun and challenges. Playing games requires knowledge, decisions, and creativity. Thus, games is consistently a field for demonstrating and examining the ability of artificial intelligence. There are many types of games, for example, shooting games, puzzle games, car racing games, and so on. Collectible card games (CCG) are a type of game in which players collect and utilize cards to achieve the goal of game (e.g. defeat the opponent). Magic: the Gathering [1] and Hearthstone [2] are two representative CCGs. This research is inspired by contests held in recent IEEE Congress on Evolutionary Computation (CEC) and IEEE Conference on Games (COG). The goal of contest is to design a game-playing agent for a recently released CCG called Legends of Code and Magic (LoCM).

LoCM is a two-player turn-based CCG. Like a typical CCG, there are a variety of cards in LoCM. Cards may have different attack points, defense points, and mana cost. They may also have special abilities that could turn the game around. In each turn, a player can use multiple cards to attack the opponent and the opponent's cards. A player wins the game when he/she decreases the opponent's life to zero. Detailed rules can be found in [3][4]. One feature of LoCM is its draft phase. In most CCGs, each player prepares a card deck and plays multiple games with the same deck. However, in LoCM players need to select cards to build the deck in the draft phase of each game. More specifically, there are 30 turns in the draft phase, and players need to choose one card from three random candidate cards in each turn. A player cannot know which card is chosen by the opponent. The deck is randomly shuffled before entering the battle phase, and hence the order in which the cards appear to be available is unknown. Each card can be used only once in the battle phase. Building a good card deck is certainly important to win the game.

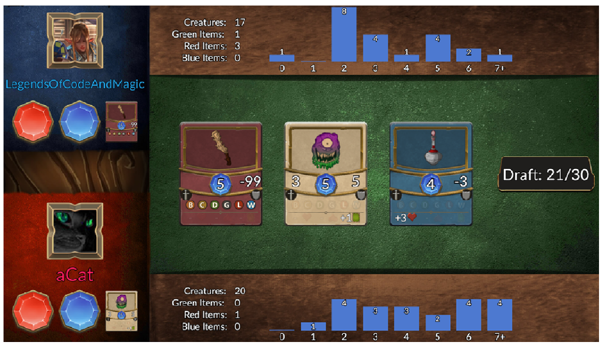


Fig. 1. Draft phase in LoCM [4]

Genetic algorithms (GA) are a kind of search algorithm and have shown successful applications in many fields. As the name reveals, GAs solve problems by a process that imitates the evolutionary process in nature. Creatures evolve to fit the environment, and solutions evolve to improve the objective function. As long as we encode or represent solutions in a proper form, GAs can search for good solutions in the encoding space. GAs have been applied to design game-playing agents in computer games. A typical way is to use GAs to search for good parameter values of rule-based agents [5]–[7].

In this paper, we focus on the deck building part of the game-playing agent for LoCM. As mentioned, the main task of deck building is to choose one card from three candidate cards in each turn in the draft phase. Our strategy is to calculate scores of cards and then pick up the card with the highest score. Thus, the scoring function is the key. We examine three versions of GA, each of which formulates the scoring function in a different way. We conduct experiments to compare their performance and investigate their pros and cons.

The rest of this paper is organized as follows. Section II reviews related studies. Section III details the proposed deck-building genetic algorithm (DBGA). Experiments and results are presented in Section IV. Conclusions and future work are given in Section V.

II. Related work

A. Legends of Code and Magic

A LoCM game consists of two phases – the draft phase followed by the battle phase. As mentioned in the previous section, a player has to select cards in 30 turns in the draft phase to build a card deck. Then, the card deck is randomly shuffled, and the battle phase begins. In each turn of the battle phase, one card in the deck becomes available on hand for summoning. (Some cards can bring extra card draws in the next turn after they are summoned.) A player can summon hand cards to the battle field and use cards on the field. Summoning cards consumes mana, and cards may have different mana cost. At the first turn, player 1 has 1 mana point and player 2 has 2 mana points. Both players get one more mana point in each of the next eleven turns. Starting from the 12th turn, player 1 (resp. player 2) has 12 (resp. 13) mana points in each turn. There are two lanes in the battle field. A player needs to decide which lane to put the summoned creature card. A creature cannot attack opponent creatures in the different lane. An item card can be used to apply its effect upon creatures on the field and both players. Each player summons and uses cards to attack the opponent and the opponent's cards. Each player has 30 health points (HP) initially, and the player whose HP becomes 0 loses the game.

The LoCM game was released in 2018 and is a quite new game. Thus, there are not many studies on LoCM. Kowalski and Miernik [8], the authors of LoCM, applied an evolution algorithm (EA) to solve the deck building problem. The chromosome represents the scores of cards. In the draft phase of each game, 60 out of 160 cards are randomly selected in the candidate pool. Then, in each of 30 turns, three cards are randomly selected from the pool to be chosen by players. In other words, at most 60 different cards appear in a game. To utilize this knowledge, Kowalski and Miernik proposed the concept of active genes, which refer to the scores of cards that appear in a game. Only active genes are considered during crossover and mutation. In 2021, the authors studied three chromosome encoding schemes – real vector, binary tree, and n-ary tree [9]. They recommended to use the simple real vector representation when the computational budget is not high and to transform the real vector to tree representation when the computational budget is high. They also investigated the opponent used in the evaluation of individuals. They found that self-play is better than the use of a fixed opponent.

Witkowski *et al.* [10], who won the champion of the COG 2020 LoCM contest, applied harmony search [11] for the draft phase and Monte-Carlo tree search [12] in the battle phase. They also tried several prediction methods to guess the opponent's card deck. The predict MCTS performed quite well. Vieira *et al.* [13] combined neural networks and reinforcement learning for both drafting and battling. For drafting, the neural network takes card attributes of current choices and past picks as input and then outputs probabilities of choosing the three candidate cards. For battling, the other neural network takes players' information and the attributes of all visible cards on the board as input and then outputs the probabilities of 163 actions. Later they [14] focused on drafting and used deep neural networks. Their three versions of drafting methods were the best three out of eight tested methods. They also participated in the COG 2020 LoCM contest, and their agent, Reinforcement Greediness (hereafter called ReGr), received the fourth place. Le developed the agent Coac [15], which won the first place in the LoCM contests in COG 2019 and CEC 2019 and the second place in COG 2020. It selects cards based on self-defined card scores and plays cards by the minimax algorithm. Montoliu *et al.* [16] used online evolution planning to decide the actions in the battle phase. The action sequences were evaluated based on a heuristic function, and the parameters in the heuristic function was tuned by an n-tuple bandit EA.

B. Other Card Games

We also review some studies that focus on deck building in other popular CCGs including Hearthstone and Magic: the Gathering (MTG). Deck building in Hearthstone aims to find a set of 30 cards from a large number of cards. The card deck must follow the rules such as no duplicate card and at most one Legendary card. García-Sánchez *et al.* [17] applied an EA, in which each individual represents a card deck. To deal with the feasibility of the card deck, their fitness function considered not only win rates but also the number of violations of the deck rules. In their later work [18], they proposed a smart mutation that only swaps two cards whose casting costs differ by no more than one point. This was inspired by the human experience that a very cheap card is unlikely replaced by a costly card during deck building. The concept of deck archetype was also mentioned. Common archetype include Aggro, Combo, and Control [19]. Stiegler *et al.* [20] proposed a utility system to evaluate cards according to several metrics including cost-effectiveness of cards, synergies between cards, mana curves, and so on.

Bjørke and Fludal [21] addressed deck building in MTG. In MTG, a card pool is first determined by opening six (random) card packs and then players build their deck by taking only cards in the pool. The authors applied a GA, and an individual in the GA also represents a card deck. They proposed a special crossover to deal with the feasibility of card deck. The cards in the two parents are combined and sorted. Then, one offspring takes the cards at odd positions and the other offspring takes those at even positions.

III. Deck-Building Genetic Algorithm (DBGA)

In this research we focus on building a card deck in LoCM. We use GA to search for a scoring function so that a game agent can use it to choose cards. We adopt three encoding schemes, which correspond to three forms of scoring functions. Details are presented in the following subsections.

A. Chromosome Encoding Schemes

(1) Direct scoring [8]

The direct scoring encoding scheme is an intuitive scheme. It aims to search for a proper score directly for each card so that the game agent can select the card with the highest score. Since there are 160 cards in LoCM, the length of chromosome is 160. Fig. 2 illustrates the chromosome. Each gene SC_i refers to the score of the card of id i.

SC_1	SC_2	SC_3		SC_{158}	SC_{159}	SC_{160}
19	2	12	…	7	11	6

Fig. 2. Direct scoring encoding

(2) Attribute scoring

In LoCM, cards are distinguished by their attributes and special abilities: three attributes are mana cost , attack, and defense; six abilities include Breakthrough, Charge, Drain, and so on. It is reasonable to compare cards based on their attribute values and the owned abilities. Some players may think of attack as a more important attribute, and some players may think of Drain as a critical ability. Thus, our second scheme encodes the weights of three attributes and six abilities, as shown in Fig. 3.

w_{atk}	w_{def}	w_{mana}	w_B	w_C	w_D	w_G	w_L	w_W
6	-4	-3	-2	1	-1	4	1	-8

Fig. 3. Attribute scoring encoding

Given a chromosome of this encoding scheme, we can calculate the score of a card by linear weighted sum of gene values (weights) and the corresponding attribute/ability. The weights of ability are counted only if the card owns the ability. For example, if a card owns Breakthrough and Drain abilities, its score is calculated by

$$score = w_{\text{atk}} \cdot atk + w_{\text{def}} \cdot def + w_{\text{mana}} \cdot mana + w_{\text{B}} + w_{\text{D}}. \quad (1)$$

After calculating the scores of cards, the card with the highest score is chosen.

(3) Attribute-Pattern scoring

In addition to card attributes and abilities, advanced players would consider building a card deck with a variety of cards. For example, they choose some lower-cost and weaker cards and some higher-cost and stronger cards. The third encoding scheme extends the second scheme by adding a group of genes that stand for the desired number of cards with different mana cost. Fig. 4 illustrates a chromosome of this encoding scheme.

n_1	n_2	n_3	n_4	n_5	n_6	n_{7+}
5	5	3	4	0	2	1

w_{atk}	w_{def}	w_{mana}	w_B	w_C	w_D	w_G	w_L	w_W
6	-4	-3	-2	1	-1	4	1	-8

Fig. 4. Attribute & Pattern scoring encoding

In Fig. 4, the gene n_1 refers to the desired number of cards with mana cost 1 , n_2 refers to the desired number of cards with mana cost 2, and the remaining n_i genes are interpreted in a similar way. Given a chromosome of this encoding, we calculate the urgency of each candidate card first. Assume it is the t^{th} ($1 \leq t \leq 30$) turn in the draft phase. Let d_i denote the number of cards with mana cost i in the current card deck. Assume the mana cost of a candidate card is i. The urgency of this card is calculated by

$$urgency(i) = n_i/30 - d_i/t. \quad (2)$$

Simply speaking, a positive *urgency* means that the number of cards in the current deck does not reach the desired number and a negative *urgency* means that the number of cards exceeds the desired number. For example, let n_i be 6 and t be 10. If the card deck already contains 2 cards ($d_i = 2$) with mana cost i, *urgency* is zero (6/30 – 2/10 = 0). Having 2 cards on hand at the 10th turn exactly matches the goal that aims to have 6 cards after 30 turns. If we have only 1 card, the number of card is not enough and *urgency* is positive; if we have 3 cards, *urgency* is negative. If there is at least one card with positive *urgency*, then we calculate the *score* of these urgent cards by eq. (1) and select the card with the highest *score*. If there is no urgent card, we select the card with the highest *score* from the non-urgent cards.

B. Chromosome Evaluation (Fitness Assignment)

The goal of our DBGA is to find a good scoring function to help game agents to choose cards and build a card deck to beat the opponent. In other words, the final goal is to win the game. Thus, the fitness of a chromosome is assigned by the win rate of the resulted game agent. Since DBGA only focuses on deck building, we take the battle part of some existing agents (which will be described in detail in Section IV) to make a complete agent. The opponent agent is Coac, which is a strong competitor. Recall that in LoCM, candidate cards in the draft phase are selected randomly and the card deck is randomly shuffled before the battle phase. Due to the randomness, we need to simulate more than one game to estimate the win rate of an agent. Considering the estimation accuracy and required computation time, we take the win rate over 50 games. Results of games are obtained by running the official game simulation engine [22]. Note that we fix the seeds of random number generator so that all chromosomes are evaluated by the same 50 games.

C. Whole Algorithm

The proposed DBGA follows the process flow of a typical GA. The initial solutions are generated randomly. To do crossover, one parent is selected by roulette wheel selection and the other is selected randomly. The hybrid of two selection strategies attempts to utilize genes of good solutions and avoid converging too quickly. Arithmetic crossover is applied to the chosen parents to generate two offspring. Let x_{ij} denote the j^{th} gene of parent X_i, and let y_{ij} denote the j^{th} gene of offspring Y_i. The offspring Y_1 and Y_2 are produced by the following equations, respectively.

$$y_{1j} = \alpha \cdot x_{1j} + (1 - \alpha) \cdot x_{2j} \quad (3)$$
$$y_{2j} = (1 - \alpha) \cdot x_{1j} + \alpha \cdot x_{2j} \quad (4)$$

Each offspring then undergoes mutation with probability p_{m}. If mutation is to be carried out, each gene keeps its original value or is changed to a random value in the prespecified range with equal probability.

After generating N_{pop} offspring, the environmental selection picks up $N_{\text{pop}} \cdot p_{\text{e}}$ (p_{e} is the proportion of elites) best solutions and $N_{\text{pop}} \cdot (1 - p_{\text{e}})$ random solutions from the union of the original population and the offspring. The above steps repeat for N_{gen} generations. Values of algorithm parameters will be given in Section IV.

IV. Experiments and Results

A. Experimental Setting

In the following experiments, we examine the performance of three versions of proposed algorithms. We call the Deck-Building GA with direct scoring, attribute scoring, and attribute-pattern scoring as encoding schemes by DBGA-D, DBGA-A, and DBGA-AP, respectively. Parameter setting of each version of DBGA is listed in Table I. We gave a larger population size to DBGA-D since its chromosome is longer and the search space is larger.

The best solution found by each version of DBGAs over multiple runs was tested by playing against existing agents in the COG2020 LoCM contest for 5000 games. (For every pair of competing agents, an agent played as player 1 for 2500 games and as player 2 for the other 2500 games.) All DBGAs were implemented by Python programming language, and the computing environment is Windows 10. Simulation of one game took about 1.3 second. A single run of DBGA-D and DBGA-A took about 23 hours and 11 hours, respectively, under the use of concurrent.futures module in Python.

TABLE I. ALGORITHM PARAMETER SETTING

Parameter	Value
N_{pop} (population size)	100 for DBGA-D 50 for DBGA-A and DBGA-AP
Range of gene values (gene values are integers)	[1, 20] & [1, 100] for DBGA-D [-10, 10] for DBGA-A [-10, 10] for weights in DBGA-AP [0, 10] for desired number of cards in DBGA-AP
N_{gen} (generation number)	50
α (parameter of arithmetic crossover)	0.3
p_m (mutation probability)	0.1
p_e (proportion of elites)	0.7

B. *Performance of DBGA-D (Direct Scoring Encoding)*

In the first experiment we examined the performance of DBGA-D. DBGA-D aims to find a set of proper scores for all cards directly. Since DBGA-D focuses only on building the card deck, we took the battle part of the agent Coac to act in the battle phase of the game. We chose Coac because it won the second place in the COG 2020 LoCM contest and it is easy to extract the battle part of its source code. We tested two versions of DBGA-D, one with the range of gene values in [1, 100] and the other with the range in [1, 20]. Each version of DBGA-D was run with the setting in Table I for three times, and the best solution over three runs was used to play against the full version of Coac. We want to know whether DBGA-D can improve the deck-building part of Coac. We also took a random search algorithm as the baseline. The random search (RS) algorithm generated 5100 random solutions (Each solution was a 160-D integer vector with random values in the range [1, 100].), and the best one was taken for performance comparison. The number 5100 is equal to the total number of solutions generated by DBGA-D ($N_{pop}\times(N_{gen}+1)$ = 100×(50+1) = 5100). RS was compared with DBGA-D to check whether the evolutionary process works.

The LoCM game is a two-player game. We fixed Coac as player 2 and took four agents as player 1: Coac, the best solution obtained by RS, and the best solutions obtained by two versions of DBGA-D. Each pair of two players played 250 games and the win rate of player 1 was recorded. This repeated for 10 times, and we recorded 10 win rates for each of the four tested agents. These win rates are drawn by box plots in Fig. 5.

First, we can see that the win rate of Coac as player 1 against Coac as player 2 is around 60%. It may imply the advantage of player 1 over player 2 in the LoCM game. (In each turn, player 1 acts earlier than player 2.) Second, the win rates of the other three agents as player 1 are not greater than 40%. Both versions of DBGA-D have higher win rates than RS does. It confirms that the evolutionary process is able to find better solutions than searching randomly. The medians of the win rates of the two versions of DBGA-D are very close. The difference in the performance between these two versions is that the version of DBGA-D using a smaller range of genes performs more stably. Overall speaking, the solutions by RS and DBGA-D do not perform satisfactorily. This inspired us to think about the second encoding scheme – attribute scoring.

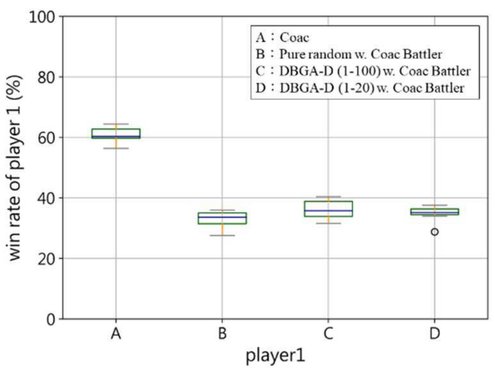


Fig. 5. Performance verification of DBGA-D

C. *Performance of DBGA-A (Attribute Scoring Encoding)*

In the second experiment we examined the performance of DBGA-A, which aims to find a scoring function based on card attributes and special abilities. As what we did in the first experiment, we ran DBGA-A with the battle part of Coac to get the best solution over ten runs. Then, this solution (agent) played against Coac for 2500 games. We recorded the win rates every 250 games and drew the box plot.

Fig. 6 shows the box plot of the win rates of DBGA-A and two versions of DBGA-D. We can see that the DBGA-A improves the median of win rates from 36% to 42% and improves the highest win rate from 40% to 48%. It means that searching for a proper attribute-based scoring function is easier than directly searching for proper scores of cards. Note that we halved the population size of DBGA-A to 50 since the chromosome length of DBGA-A (9) is much smaller than that of DBGA-D (160). Thus, the attribute scoring encoding is better than the direct scoring encoding in terms of both solution quality and computational efficiency. However, the win rate of DBGA-A with Coac battler is lower than 50%, which means that the original Coac is still better. We did more investigations, and the results will be presented in the next two sub-sections.

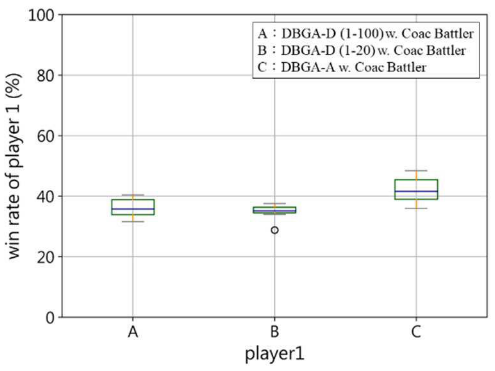


Fig. 6. Performance verification of DBGA-A w. Coac Battler

In order to know whether the idea of DBGA-A works generally, in this experiment we also tested DBGA-A with the battle part of another agents, ReGr in the COG 2020 LoCM contest. ReGr was the fourth place in the contest. In addition, we created an agent by combining the draft part of Coac and the battle part of ReGr. This agent was used to check how good DBGA-A is. Three agents (the original ReGr, DBGA-A with ReGr battler, and Coac draft with ReGr battler) played as player 1 against Coac for 2500 games and as player 2 for another 2500 games. Fig. 7 shows the box plot of win rates of tested players against Coac. In Fig. 7, box plots A and B are the win rates of ReGr as player 1 and player 2 respectively. The remaining box plots C to F are interpreted in the same way.

First, we can see that the median of win rates of the original version of ReGr is about 40% and 24%. Replacing the draft part of ReGr by the draft part of Coac, the win rate improves to 50% when the agent played as player 1, but the win rate slightly decreases when the agent played as player 2 (compare box plots B and D). By using DBGA-A for drafting, the win rate of ReGr improves to 48% (box plot E) and 32% (box plot F), respectively. The results showed that DBGA-A is able to find a better deck-building strategy for ReGr and the strategy works no matter the agent plays as player 1 or 2.

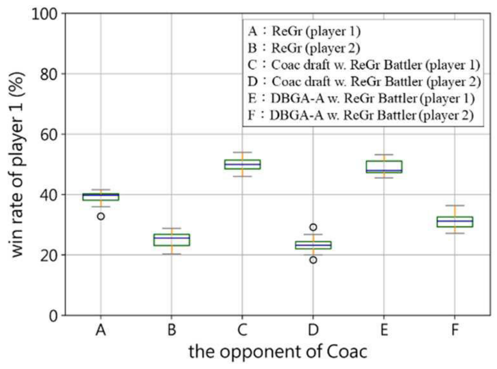


Fig. 7. Performance verification of DBGA-A w. ReGr Battler

D. *Analysis of Deck Patterns*

In the previous experiment, we found that DBGA-A can significantly improve the win rate of an existing agent ReGr. However, the win rate of the improved ReGr is still below 50% when playing against Coac. To find out why DBGA-A cannot help ReGr to beat Coac, we analyzed the patterns of card decks built by DBGA-A and Coac.

We collected the card decks of DBGA-A and Coac from 250 games. Then, we calculated the average number of cards in the decks with respect to the mana cost . Cards with mana cost seven or higher are counted together. Fig. 8 shows the distribution curves of the average number of cards in the decks built by DBGA-A and Coac. We found that DBGA-A tends to choose cards with low mana cost but Coac chooses more cards with high mana cost. This observation inspired us to devise the third encoding scheme – attribute-pattern scoring.

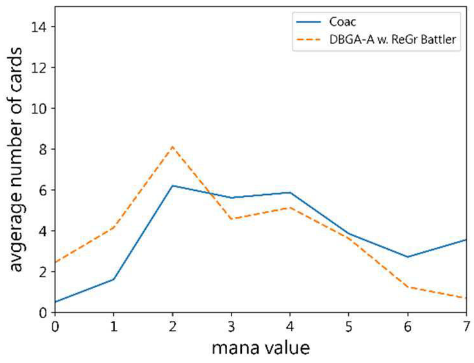


Fig. 8. Distribution curves of the average number of cards (DBGA-A vs. Coac)

E. *Performance of DBGA-AP (Attribute and Pattern Scoring Encoding)*

In the third experiment we examine the performance of DBGA-AP, which aims to find a scoring function based on card attributes and the desired numbers of cards that consume different mana. Since we found that DBGA-A could improve the win rate of the agent ReGr, we continued to test DBGA-AP on ReGr. We ran DBGA-AP with the battle part of ReGr to get the best solution over ten runs. Then, this solution (agent) played against Coac for 2500 games. We collected the win rates every 250 games and drew the box plot in Fig. 9. Although DBGA-AP builds better card decks for ReGr (the win rate is higher than that of the original ReGr), DBGA-AP is not as good as DBGA-A.

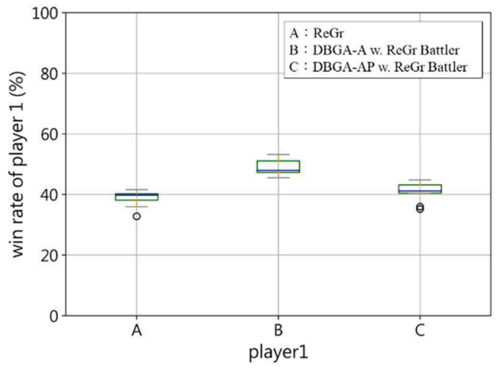


Fig. 9. Performance verification of DBGA-AP w. ReGr Battler

After seeing the results in Fig. 9, we checked the distribution curves of cards chosen by DBGA-AP, as shown in Fig. 10. The two curves are closer than the two curves are in Fig. 8. In other words, DBGA-AP indeed adjusts the pattern of the card deck. So, what else can be the problem?

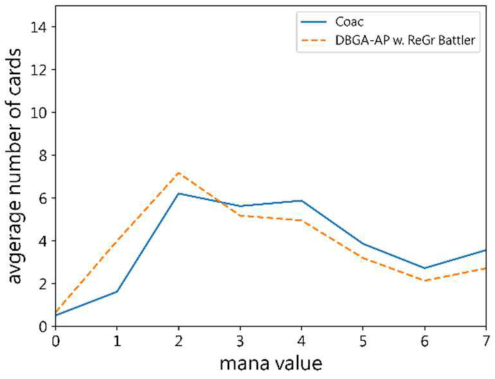


Fig. 10. Distribution curves of the average number of cards (DBGA-AP vs. Coac)

Considering the rules of LoCM, high-cost cards cannot be played in early turns of the game since players do not have enough mana. (Player 1 (resp. 2) has 1 (resp. 2) mana at the first turn, and the mana increases by 1 gradually as the turn goes until turn 12.) Noticing this difference, we analyzed the number of turns taken in the games won by DBGA-AP and that in the games won by Coac. We drew the data by box plots in Fig. 11. We can see that DBGA-AP won within fewer turns and Coac won when the battles last for more turns.

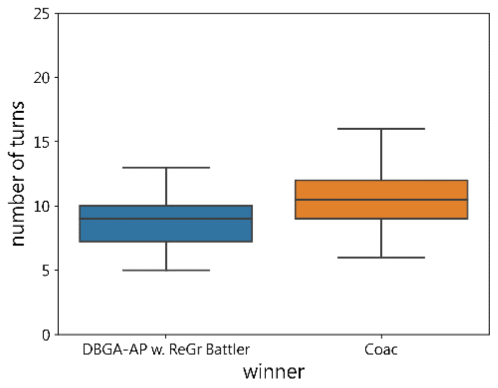
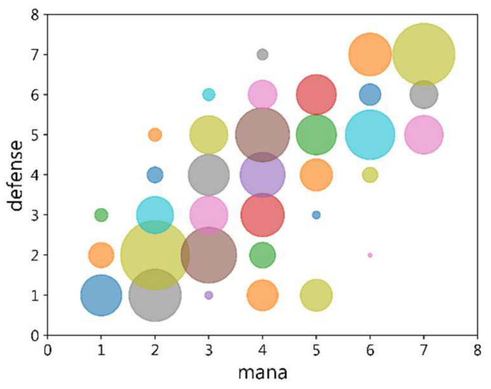
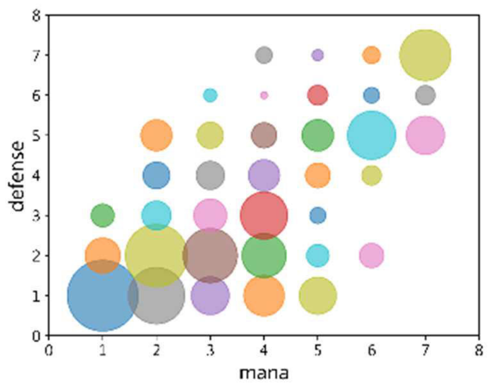


Fig. 11. The number of turns ending with a certain agent as the winner

DBGA-AP already builds the card deck with a similar pattern as that of Coac, and it also selects high-cost cards. Why does the deck obtained by DBGA-AP still gets weak when the battle goes with more turns? In addition to mana cost , we took one more card attribute into the analysis – defense. Fig. 12 shows the bubble charts of card decks built by DBGA-AP and Coac, respectively. The x-axis stands for the mana cost , and the y-axis stands for defense. The size of a bubble centered at (x, y) reflects the number of cards with mana cost x and defense y in the built deck. Comparing the two bubble charts, we found the difference between the decks of DBGA-AP and Coac. Coac selects more cards with high defense than DBGA-AP does. We think that this difference explains why Coac could win when the battle lasts longer. (Creatures with higher defense can live longer and fight back.)



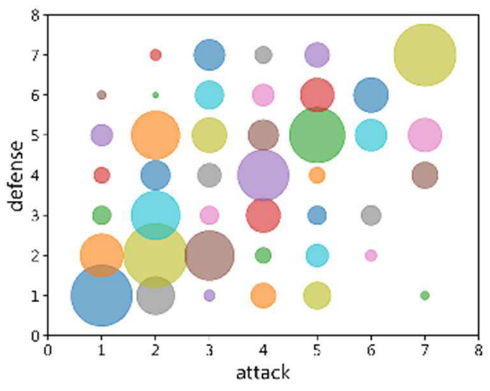
(a) Coac



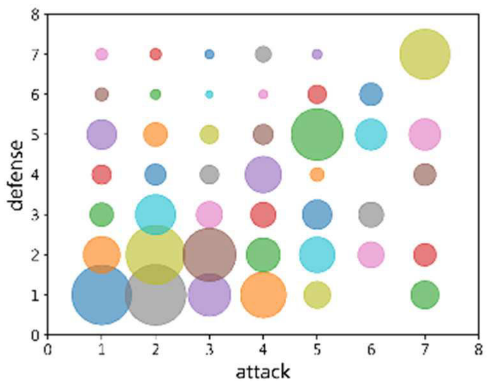
(b) DBGA-AP

Fig. 12 Bubble charts of the card deck (mana vs. defense)

As for the reason why DBGA-AP does not select enough cards with high defense, we think that attribute-based scoring is the cause. As mentioned in Section III, DBGA-AP chooses cards considering the desired number of cards first and then the aggregated score based on attribute values. The single scoring function makes DBGA-AP choose cards with similar characteristics. If the scoring function prefers cards with high attack and low defense, DBGA-AP chooses a lot of cards of this kind; if the scoring function prefers cards with low attack and high defense, DBGA-AP chooses a lot of cards of that kind. In this situation, the best way to raise the win rate is to choose a lot of similar cards with low mana cost, sufficiently high attack, and consequently low defense due to the balanced design of cards in the LoCM game. (It is unlikely to have cards that consume low mana but have high attack and high defense in a CCG.) We do not get a high win rate when we select a lot of cards with low attack and high defense; in this case, we cannot beat down the troop of the opponent. We do not get a high win rate when we select a lot of cards with high attack and high defense; these cards consume high mana and we will be beaten down in early turns of the game. Thus, it is reasonable that DBGA-AP selects more cards with low cost, medium-to-high attack, but low defense. This is confirmed by Fig. 13, which shows the bubble charts of the card deck with attack as the x-axis and defense as the y-axis.



(a) Coac



(b) DBGA-AP

Fig. 13 Bubble charts of the card deck (attack vs. defense)

The next step is to do a similar design on attack and defense as what we have done on mana. We can encode the desired number of cards with different attack values and defense in the chromosome. However, it may enlarge the search space and require much longer time to evolve the population. We leave this as a topic in our future work.

F. *Comparison with Existing Agents*

Among the three versions of DBGAs, DBGA-A performs the best. Thus, we took DBGA-A with ReGr battler as our agent and tested it by six agents in the COG 2020 LoCM contest. We ran DBGA-A as player 1 and player 2 separately and got specific scoring functions for playing as player 1 and 2, respectively. For each pair of two competing agents among the seven tested ones, the two agents played 5000 games, in which each agent plays as player 1 for 2500 games. The win rate of each agent over 5000 games is recorded. All win rates obtained by running $21 \cdot 5000 = 105000$ games are presented in Table II. The overall average win rates are presented in Table III. The DBGA-A with ReGr battler is slightly better than two agents in the contest but is still significantly outperformed by two other agents.

TABLE II. WIN RATES OF THE ROW AGENT AGAINST THE COLUMN AGENT

	Coac	Chad	DBGA-A w. ReGr	OneLanels Enough	ReGr	Base line1	Base line2
Coac	–	70.4	59.0	57.8	68.3	97.5	97.1
Chad	29.6	–	96.2	58.8	95.7	81.8	86.5
DBGA-A w. ReGr	**41.0**	**3.8**	**–**	**60.7**	**47.8**	**95.6**	**97.1**
OneLanels Enough	42.2	41.2	39.3	–	44.7	83.9	90.0
ReGr	31.7	4.3	52.2	55.3	–	95.1	95.8
Baseline1	2.5	18.2	4.4	16.1	4.9	–	39.9
Baseline2	2.9	13.5	2.9	10.0	4.2	60.1	–

TABLE III. AVERAGE WIN RATES AND RANKS

Agent	Avg. Win Rate (%)	Rank
Coac	75.0	1
Chad	74.8	2
DBGA-A w. ReGr	**57.7**	**3**
OneLanelsEnough	56.9	4
ReGr	55.7	5
Baseline2	15.6	6
Baseline1	14.3	7

V. CONCLUSIONS

This paper addressed the deck building procedure in a collectible card game, LoCM. We proposed to use GA to search for a good scoring function to evaluate cards and to select cards accordingly. Three encoding schemes were proposed, including direct scoring, attribute scoring, and attribute-pattern scoring schemes. We tested them with the battle part of two existing agents (Coac and ReGr) and did investigations into the pros and cons of these three encoding schemes. The results showed that searching for a scoring function based on card attributes (DBGA-A) is more effective and efficient than searching for card scores directly (DBGA-D). In addition, we should not select a lot of cards with similar characteristics. In this paper, we have made an initial attempt to consider the distribution of the number of cards with different mana cost (DBGA-AP). We found the number of cards must be balanced with respect to not only mana but also other attributes such as defense.

This research can be continued with the following directions. First, advanced encoding schemes can be tried. For example, we can take more card attributes (e.g. health change and card draw bonus) into account or use genetic programming to search for a more complicated function for evaluating cards. Second, we can consider more factors in the fitness function. Currently we only consider the win rate. We may consider the remaining life of players, the number of turns taken in games, and so on. Third, the genetic operators in our algorithm can be improved. For example, our current use of arithmetic crossover may have a bias to search for the mid part of the range of gene values. Last, simulation of games takes a lot of computation time, especially when we plan to run our DBGA based on different kinds of battle agents. Parallel genetic algorithms and techniques in the field of expensive optimization can be useful.

REFERENCES

[1] Magic: The Gathering, https://magic.wizards.com/zh-hant

[2] Hearthstone, https://playhearthstone.com/zh-tw/

[3] Legends of Code and Magic on Coding Game, https://www.codingame.com/multiplayer/bot-programming/legends-of-code-magic

[4] Legends of Code and Magic, https://legendsofcodeandmagic.com/

[5] M. Gallager and A. Ryan, "Learning to play Pac-Man: an evolutionary, rule-based approach," Proceedings of IEEE Congress on Evolutionary Computation, pp. 2462−2469, 2003.

[6] N. Cole, S. J. Louis, and C. Miles, "Using a genetic algorithm to tune first-person shooter bots," Proceedings of IEEE Congress on Evolutionary Computation, pp. 139−145, 2004.

[7] N. Böhm, G. Kókai, and S. Mandl, "An evolution approach to Tetris," Proceedings of the Sixth Metaheuristics International Conference, 2005.

[8] J. Kowalski and R. Miernik, "Evolutionary approach to collectible card game arena deckbuilding using active genes," Proceedings of IEEE Congress on Evolutionary Computation, 2020.

[9] R. Miernik and J. Kowalski, "Evolving evaluation functions for collectible card game AI," 2021.

[10] M. Witkowski, Ł. Klasinski, and W. Meller, "Implementation of collectible card game AI with opponent prediction," Master Thesis, University of Wrocław, 2020.

[11] Z. Woo, J. Hoon, and G. V. Longanathan, "A new heuristic optimization algorithm: harmony search," Simulation, vol. 76, no. 2, pp. 60−68, 2001.

[12] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," Proceedings of European Conference on Machine Learning, 2006.

[13] R. Vieira, L. Chaimowicz, and A. R. Tavares, "Reinforcement learning in collectible card games: preliminary results on Legends of Code and Magic," Proceedings of the 18th Brazilian Symposium on Computer Games and Digital Entertainment, SBGames, pp. 611−614, October 2019.

[14] R.Vieira, A. R. Tavares, and L. Chaimowicz, "Drafting in collectible card games via reinforcement learning," Proceedings of the19th Brazilian Symposium on Computer Games and Digital Entertainment, SBGames, pp. 54−61, 2020.

[15] Coac github, https://github.com/Coac/locm

[16] R. Montoliu, R. D. Gaina, D. Pérez-Liébana, D. Delgado, and S. Lucas, "Efficient heuristic policy optimisation for a challenging strategic card game," Proceedings of the International Conference on the Applications of Evolutionary Computation (Part of EvoStar), pp. 403–418, 2020.

[17] P. García-Sánchez, A. Tonda, G. Squillero, A. Mora, and J. J. Merelo, "Evolutionary deckbuilding in Hearthstone," Proceedings of the 2016 IEEE Conference on Computational Intelligence and Games, 2016.

[18] P. García-Sánchez, A. Tonda, A. Mora, G. Squillero, and J. J. Merelo, "Automated playtesting in collectible card games using evolutionary algorithms: a case study in Hearthstone," Knowlege Based Systems, vol. 153, pp. 133−146, 2018.

[19] Identifying deck archetypes, (https://tempostorm.com/articles/identifying-deck-archetypes)., 2016.

[20] A. Stiegler, C. Messerschmidt, J. Maucher, and K. Dahal, "Hearthstone deck-construction with a utility system," Proceedings of the 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA), pp. 21–28, 2016.

[21] S. J. Bjørke and K. A. Fludal, *Deckbuilding in Magic: the Gathering using a Genetic Algorithm*, Master Thesis, Norwegian University of Science and Technology, 2017.

[22] Strategy Card Game AI Competition, https://github.com/acatai/Strategy-Card-Game-AI-Competition