Evolutionary Approach to Collectible Card Game Arena Deckbuilding using Active Genes

Jakub Kowalski
Institute of Computer Science
University of Wrocław
Wrocław, Poland
jko@cs.uni.wroc.pl

Radosław Miernik
Institute of Computer Science
University of Wrocław
Wrocław, Poland
radekmie@gmail.com

***Abstract*—In this paper, we evolve a card-choice strategy for the arena mode of Legends of Code and Magic, a programming game inspired by popular collectible card games like Hearthstone or TES: Legends. In the arena game mode, before each match, a player has to construct his deck choosing cards one by one from the previously unknown options. Such a scenario is difficult from the optimization point of view, as not only the fitness function is non-deterministic, but its value, even for a given problem instance, is impossible to be calculated directly and can only be estimated with simulation-based approaches.**

We propose a variant of the evolutionary algorithm that uses a concept of an *active gene* to reduce the range of the operators only to generation-specific subsequences of the genotype. Thus, we batched learning process and constrained evolutionary updates only to the cards relevant for the particular draft, without forgetting the knowledge from the previous tests.

We developed and tested various implementations of this idea, investigating their performance by taking into account the computational cost of each variant. Performed experiments show that some of the introduced active-genes algorithms tend to learn faster and produce statistically better draft policies than the compared methods.

***Index Terms*—Evolutionary Algorithms, Collectible Card Games, Deck Building, Game Balancing, Strategy Card Game AI Competition, Legends of Code and Magic**

I. Introduction

Currently, not only classical boardgames like Chess [1] and Go [2] are used as grand challenges for AI research. It has been recently shown that such a role may be taken by modern computer games. So far presented approaches that beat the best human players in *Dota 2* [3] and *StarCraft II* [4] are one of the most spectacular and media-impacting demonstrations of AI capabilities.

The weight is put on particular game features that make designing successful AI players especially tricky, e.g., imperfect information, randomness, long term planning, and massive action space. One of the game genres containing all these game features is Collectible Card Games [5].

Recently, numerous research has been conducted in this domain, focusing mainly on development of MCTS-based agents, and creating the deck recommendation systems that will choose the right set of cards to play. The *Hearthstone AI Competition* [6], with the goal to develop the best agent for the game *Hearthstone*, [7] was organized during the IEEE CIG/COG conferences in 2018 and 2019. The *Strategy Card Game AI Competition* based on programming game *Legends of Code and Magic* (LOCM) [8], designed especially for handling AI vs. AI matches and played in the arena mode, was hosted in 2019 by IEEE CEC and IEEE COG. The *AAIA'17 Data Mining Challenge: Helping AI to Play Hearthstone* [9] was focused on developing a scoring model for predicting win chances of a player, based on single-game state data.

This work is the first approach to build a deck recommendation system for the *arena* mode. As in the other modes players can use their full collection of cards, in arena they draw a deck from a random selection of cards before every game. Such a task is characterized by an even larger domain (considering all possibilities of available choices), higher non-determinism (the additional card selection phase is non-deterministic), and harder opponent prediction.

We propose variants of the evolutionary algorithm that uses a concept of an *active gene* to reduce the range of the operators only to specific subsequences of the genotype that changes from generation to generation. In other words, our batched learning constrains evolutionary updates only to the cards relevant for the particular draft. Individuals forming the next population are no longer simply selected among the parents/offspring populations, but instead, they are merged from the specific subsets of their representatives.

We have conducted a series of experiments in LOCM to estimate the performance of multiple variants of designed algorithms and compare them with several baselines. Algorithms learn on a small sample of random drafts (available card choices) and are tested on a larger number of other drafts, using large number of game simulations to estimate the fitness value in such a highly non-deterministic environment. The results show that some of the introduced active genes variants perform better than other tested approaches, i.e., they tend to draw stronger decks from the same available card choices.

II. Background

A. Collectible Card Games

Collectible Card Game (CCG) is a broad genre of both board and digital games. Starting with *Magic: The Gathering*

This work was supported by the National Science Centre, Poland under project number 2017/25/B/ST6/01920.

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

(MtG) [10] in the early 90s, or more recent *Hearthstone* [7] and *The Elder Scrolls: Legends* (TESL) [11], a huge number of similar games have been created.

The mechanics differ between games, but basics are similar. Two players with their *decks* draw an initial set of cards into their *hands*. Then the actual play begins. A single *turn* consists of a few *actions*, like playing a card or using an onboard card. The game ends as soon as one of the players wins, most often by getting their opponent's health to zero.

A typical CCG characterizes with a large number of playable cards (over 1,000 in TESL and almost 20,000 in MtG), which causes an enormous number of possible deck compositions. This leads to even bigger in-play search space, as the player does not know the order of his next draws, the content of the opponent's deck, nor his in-hand cards. Such numbers tend to increase the amount of problems related with the imperfect information and randomness.

It is also common to observe a *metagame* level of such games. It describes the popularity of certain decks or cards. On a top-level, meta creates a possibility to compare different decks on a larger scale. Most often, it boils down to a "rock-paper-scissors" scheme, but with more possible types.

B. Related Work

The problem of creating decks that will be effective for some given meta (usually understood as a currently dominating set of opposing decks) is one of the key challenges for CCG domain [5]. In [12], the Hearthstone decks are evolved and tested for their strength via playing against a small number of predefined human-created decks. A similar task, but in a much more complicated domain of Magic: The Gathering has been approached, also via evolution, in [13]. A neural network-based approach to deckbuilding in Hearthstone has been presented in [14]. More in-depth analysis of Hearthstone deck space and the impact of various factors on the process of their evolution can be found in [15]. All the above research is focused on the *constructed* game mode, i.e., a static card selection, from unrestricted sets of available cards.

The topic closely correlated with deckbuilding is how to design the cards in CCG to be balanced [16]. In [17], evolution is used to propose changes to the cards that will result in better balance. In [18], several experiments using a modification of MAP-Elites algorithm for design and rebalancing of Hearthstone have been presented.

So far, no research has been aimed at the arena mode of CCGs. Methods for estimating card values that could be useful in arena play have not been subject to a proper investigation, although they are popular among human game players. There exist dedicated web pages and game-helping software that recommends cards (e.g., [19], [20]). The data they are based on is continuously updated and consists of a mix of expert domain knowledge, mathematical formulas, and remarks made by players on public forums.

C. Legends of Code and Magic

Legends of Code and Magic (LOCM) [8] is a small implementation of a Collectible Card Game, designed to perform AI research. Its advantage over the real card game AI engines is that it is much simpler to handle by the agents, and thus allows testing more sophisticated algorithms and quickly implement theoretical ideas.

Fig. 1. Legends of Code and Magic – in-game visualization.

All card effects are deterministic. Thus the non-determinism is introduced only by the ordering of cards and unknown opponent's deck. The game board consists of two lanes (similarly as in The Elder Scrolls: Legends), so it favors deeper strategic thinking. Fig. 1 shows the visualization in the middle of the game. Also, LOCM is based on the fair arena mode, i.e., before every game, both players create their decks secretly from the symmetrical yet limited choices. The card choices for the players are different every game, but both players have the same decisions in this phase. This is not true in arena mode in existing computer games, where every created deck is used in several games versus players that had other options to choose from. The deckbuilding in LOCM is more dynamic and, although the concept of meta is still applicable, it can be countered by the specific choices of drafts, reducing the overall strength of usual human-created top-meta decks.

The game in a slightly simplified (one-lane) form was used in August 2018 as a CodinGame platform contest, attracting more than 2,000 players (or rather AI programmers) across the world [21]. The *Strategy Card Game AI Competition* based on LOCM was hosted in 2019 by IEEE CEC and IEEE COG [8].

III. Methodology

A. Problem Specification

Consider two players, A and B, playing in a constructed mode. Before every game, they have to choose their decks, i.e., subsets of available cards additionally fulfilling some game-specific constraints (e.g., no more than three copies of the same card). Then, A and B play against each other, using their playing algorithms and their chosen decks (randomly shuffled).

Thus, the goal of A is to choose deck and algorithm such that it performs best over every possible combination of the opponent's choices. (In practice, as the number of such combinations is huge, a much smaller set of meta opponent decks is considered, as they are well-performing and have a higher chance of appearance.)

However, considering the arena mode, the task becomes slightly different. Now, the player A has to build his deck given a set of choices in the so-called *draft* phase. Usually, it consists of turns in which A has to pick one of randomly given cards, and he is not aware of the future options. Thus, the player's goal is to choose the best draft strategy, i.e., such that performs best for every draft options and every possible opponent.

In the deckbuilding problem, we fix the algorithms used by the players, treating them as given, and focus on optimizing the policy of choosing the cards. Either a static as in constructed mode, or dynamic, depending on the given draft options, in arena.

1) Domain: Consider a set of all possible cards in the game $\mathbb{C}$. For Legends of Code and Magic, $|\mathbb{C}| = 160$. During the draft phase, a player collects 30 cards for his deck in turns, in each turn choosing one of 3 given cards. Thus, given that all draws are independent and a card cannot be repeated within a single turn, the number of possible drafts for LOCM is $(160 \times 159 \times 158)^{30} \approx 1.33 \times 10^{198}$.

During the draft, a player knows his previous decisions but is unaware of future choices. Thus, when learning the best draft policy, we search for the best function from the following domain (simplified to repeating cards):

$$(\mathbb{C} \cup \{\perp\})^{30} \times \mathbb{C}^3 \to \{1, 2, 3\},$$

where $\perp$ denotes a choice that is yet to make (remaining draft turns). We assume the cards to choose from are ordered, so it is enough to pick the position of the card.

This task can be significantly simplified by discarding the information about previous choices (which removes, e.g., taking to account card synergies or controlling a mana curve), and reduces the domain of possible draft strategies to

$$\mathbb{C}^3 \to \{1, 2, 3\}.$$

In this work, for practical reasons, we represent a draft policy as the pure card-value assignment: in each turn, the card with the highest value is chosen. Such an approach is popular as a base for the, e.g., Hearthstone arena helpers [19], [20]. It also makes the encoding of the genome easy and relatively small (vector of $|\mathbb{C}|$ real numbers), while still providing a relatively good estimation of real-card value given that learning is performed with respect to fixed playing algorithms.

2) Fitness Function: The difficulty with estimating the quality of card selection policy comes from the multiple sources of simplifying assumptions and randomness. Even approximated fitness functions, using fixed playing algorithms, are flawed not only because of unknown distribution of opponent strategies, but also the nondeterminism in the games themselves, causing the value of the function to be less reliable.

In such a case, a reasonable approach is to combine simulation-based estimations with numbers. First, sample the largest possible portion of possible drafts. Second, estimate fitness based on the performance of candidate policies on those sample drafts, using a large number of simulated games.

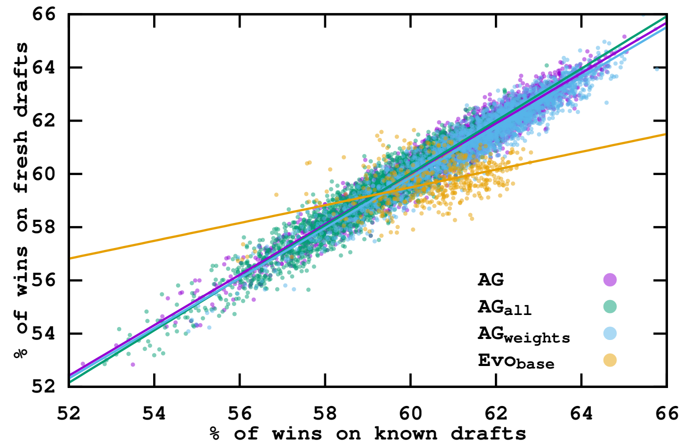

Fig. 2. Correlation between performance during training (x axis) and evaluation (y axis). All algorithms were trained on 100 drafts and evaluated on 250 drafts. A single evaluation consisted of 250 games against two baselines and three random strategies using the random player. Clearly, Evo_{base} generalizes worse than any of AG algorithms, which performance stays at the same level.

To be sound, this approach requires that the performance on a low number of training samples correlate with the performance on a larger set of unseen test samples. To ensure this is true, we performed additional tests on draft strategies of various quality and obtained from various sources. The results, presented in Fig. 2, confirms such a correlation.

B. Simple Baselines and Measuring Computational Cost

To compare the algorithms fairly, we introduce a cost measure $\mathcal{C}$, which is equal to the number of simulated games required to compute the solution. This is more accurate than simple generation-based comparison, as the total cost of calculating one generation may differ between two algorithms.

We introduce the following notation: n is the size of a population, d_t is the number of random drafts used to train the solution, s_g is the number of games played when comparing two individuals.

To provide a simple baseline for the quality of evolution, we use two types of randomly generated solutions. First, called $Random_{all}$, creates n random individuals and plays s_g games (with side change) on every training draft between every pair of the individuals. An individual which won the most games is the solution. The computation cost of this method is:

$$\mathcal{C}_{Random_{all}} = n \times (n-1) \times s_g \times d_t.$$

The second random baseline, called $Random_{tournament}$, randomizes n individuals and performs a tournament to select the best representative. To determine the winner of each matchup, s_g games on every training draft are played. Thus the computation cost is:

$$\mathcal{C}_{Random_{tournament}} = (n-1) \times s_g \times d_t.$$

We also use two precomputed card orderings, established by some of the top participants of the one-lane CodinGame LOCM contest [22]. Thus, they have no computation cost. We call this strategies $Contest_1$ and $Contest_2$.

C. Baseline Evolutionary Algorithm

The goal of the evolution is to improve the quality of draft policies encoded as chromosomes. A straightforward approach is to use all available training data in each generation.

A genotype is represented as a constant-size vector of doubles. Every gene (from 1 to $|\mathbb{C}| = 160$) encodes a priority of the associated card (estimated card value). Thus, during the draft phase, a card with the highest priority is chosen each turn.

As before, let n be the size of the population, and d_t the number of available training drafts. The schema of the baseline evolutionary algorithm Evo_{base} goes as follows.

The population is initialized with the random values between 0.0 and 1.0. To evaluate the population, we play s_g games between every two individuals for each of d_t drafts (with side change after half of the games). Then, we use tournament selection (best of random 4 individuals) to select $\frac{n}{2}$ parents based on the number of wins. The standard uniform crossover is used. Mutation rate $m = 0.05$ is the probability of changing each gene into a new random number. The elitism size is 2. Thus, to compute g generations, this algorithm has to play $\mathcal{C}_{Evo_{base}}$ games, where

$$\mathcal{C}_{Evo_{base}} = n \times (n-1) \times s_g \times d_t \times (1+g).$$

IV. Active Genes Algorithm

Evaluating each generation using all of the available test drafts, while being the most robust, is also time-consuming and may force the evolution to be insubstantial within the constrained computational budget.

Alternatively, we propose an approach where each generation is responsible for learning how to play only one of the available test drafts. Such a method allows not only evolving more generations within the same budget but also observing a more detailed influence of single genes (cards) on the performance of the agents in particular scenarios (drafts). To emphasize the benefit of this gene-to-draft-to-outcome correspondence even more, and partially make up for the loss of generality it creates, evolutionary operators will be applied selectively, differently in each generation.

We say a gene is *active* in a given generation if the card it encodes appears within the draft choices. For the sake of evaluation of this generation, all the other genes are considered irrelevant. Thus, in this generation, we can perform crossover and mutation only on the active genes. Moreover, this will prevent the destruction of the so-far gained knowledge encoded in the inactive genes, which has already proven itself through the previous generations.

The major drawback of this approach is that we lose a uniform metric that can be used to compare parents and children across the generations. Thus, to select $\frac{n}{2}$ pairs of parents, we run actual games to test how they perform on the current generation draft. For each parent we perform a tournament of size $s_{tSize} = 4$, playing $s_{tGames} = 10$ games each round. Selected parents create offspring in the same manner as in the Evo_{base} algorithm (uniform crossover, constant mutation rate).

Now, we calculate fitness values for the offspring population. This is done in s_r rounds, where in each round we score the population by the so-far wins and then play s_g games (with side change) between consecutive pairs in order.

The remaining part, selection, is in our approach substituted by a *merge* operation. To create each of n individuals of the next population, we use a fitness-based roulette to select a child from the offspring, and a parent from the parents' population. For parents, the fitness values of their generation are used.

We investigated three variants of the merging procedure. First, called AG, uses active genes most straightforwardly. The resulting individual contains values of the active genes copied from the child and inherits the rest from its parent. Thus, the newest knowledge is considered the most important.

Alternatively, $AG_{weights}$ is the variant of AG that uses the weighted sum instead. Such a variant aims to be more conservative and improve individuals gradually, preserving the already gained knowledge even more. The proportion we found working is 0.75 of the parent gene and 0.25 of the child gene, instead of the original 0 and 1 in AG.

For comparison, we also test AG_{all} variant, which is based on the same scheme but does not take advantage of active genes (i.e., it treats all genes as active). Instead, it discards the parent, and only the child gene values are copied.

Essentials of the pseudocode for these algorithms are presented in Fig. 1. Here, we assume that $g = d_t$, which is in align with most of the conducted experiments. More sophisticated variants, using the same pseudocode but breaking this assumption, are described separately in Section V-C. The main procedure, Evolve, progresses through each generation learning d_t drafts, creating random drafts using RandomDrafts, calculating the children population using CreateOffspring, and merging them, as described above, using MergeAll. Depending on the variant, MergeOne selects an appropriate behavior, where Lerp is the linear interpolation function. The Score procedure simulates a single game and updates the scores (win counts) accordingly.

Thus, the cost of g generations using these algorithms is:

$$\mathcal{C}_{AG} = \mathcal{C}_{AG_{weights}} = \mathcal{C}_{AG_{all}} = \\ n \times g \times (s_{tSize} \times (s_{tSize} - 1) \times s_{tGames} + s_r \times s_g \times d_t)$$

V. Experiments

Source code for the described experiments is available in a public GitHub repository [23]. All performed experiments were run on a single CPU-Optimized DigitalOcean droplet with 16 GB of RAM and 8 standardized vCPUs.

A. Algorithm Comparison

We compared the overall results obtained by the $Contest_1$, $Contest_2$, $Random$, $Random_t$, Evo_{base}, AG, AG_{all}, and $AG_{weights}$ algorithms. To ensure the robustness of our approach, we have tested two playing strategies – random and greedy. We did not test more advanced strategies due to the computational time constraints.

Algorithm 1 Active genes algorithm variants pseudocode.

```
procedure EVOLVE(options)
    old ← RANDOMPOPULATION(n)
    for generation ← 1, g do
        drafts ← RANDOMDRAFTS()
        new ← CREATEOFFSPRING(drafts, old)
        old ← MERGEALL(old, new, drafts, options)
    return old

procedure CREATEOFFSPRING(drafts, old)
    new ← EMPTYPOPULATION()
    for individual ∈ new do
        parents ← SELECTPARENTS(drafts, old)
        children ← CROSSOVER(parents)
        individual ← MUTATE(children)
    SCOREPOPULATION(drafts, new)
    return new

procedure SELECTPARENTS(drafts, population)
    tournament ← SAMPLE(population, s_tSize)
    for all draft ∈ drafts do
        for all a ∈ tournament do
            for all b ∈ tournament do
                if a ≠ b then
                    for game ← 1, s_tGames / 2 do
                        SCORE(a, b, draft)
                        SCORE(b, a, draft)
    return BEST2BYSCORE(tournament)

procedure SCOREPOPULATION(drafts, population)
    for round ← 1, s_r do
        SORTBYSCORE(population)
        for all draft ∈ drafts do
            for i ← 1, 3, 5, ..., n do
                a, b ← population[i], population[i + 1]
                SCORE(a, b, draft)
                SCORE(b, a, draft)

procedure MERGEALL(old, new, drafts, options)
    merged ← EMPTYPOPULATION()
    for all individual ∈ merged do
        a, b ← ROULETTE(old), ROULETTE(new)
        individual ← MERGEONE(a, b, drafts, options)
    return merged

procedure MERGEONE(new, old, drafts, options)
    if AG_all then
        cardIds ← ALLCARDIDS
    else
        cardIds ← CARDIDSIN(drafts)
    merged ← CLONE(old)
    for all id ∈ cardIds do
        old[id] ← LERP(new[id], old[id], options.weight)
    return merged
```

The first one uniformly picks a random action sequence, while the latter selects the best actions, one at a time, according to a material heuristic. A complete summary of their relative performance is presented in Tables I and II.

While the competition-based heuristic $Contest_1$ gains fewer wins than loses, $Contest_2$ seems to be the second most robust choice from considered strategies. What is interesting, during the one-lane LOCM contest [22] $Contest_1$ was established as the meta, outperforming other solutions, including $Contest_2$.

Scores of both random players are worse than those obtained by any evolution-based approach, which is a clear indication that the learning process gives a definite advantage.

Additionally, on Fig. 3, we visualized the process of evolution for Evo_{base}, AG, AG_{all}, and $AG_{weights}$. For each of the presented algorithms, the best five individuals from each generation played 50 games on 250 random drafts against $Contest_1$, $Contest_2$, and three random individuals. The results reported on this chart are higher than in the tables mentioned before, as the random opponents tend to be less skilled on average.

Surprisingly, the performance of the straightforward evolution Evo_{base} is below 50% when using the random player. Also, the correlation between performance on the training and evaluation drafts is the worst (yellow line on the Fig. 2). Each of its generations took significantly longer to compute, so only a few of them can be finished within the assumed computation budget. It might be the case that their number is too low to observe learning. Nevertheless, a few non-exhaustive experiments we had additionally performed showed that the score does not raise significantly even with a far greater computational budget or using different parameters. This variant generates average solutions during the first generation that it cannot further improve.

As expected, AG_{all} performs poorly. Using only offspring genes results in constant forgetting, which is visible in its evolution process. On the contrary, the remaining active genes-based approaches, AG and $AG_{weights}$, learn step-by-step from low scores. Both need about four times the cost of the initial Evo_{base} generation to start performing better, but the process does not finish there. They continue learning, which supports the assumption of the advantage of batched learning and selective genetic operators. The variant with weighted merge achieves significantly better results versus all opponents, especially using the greedy playing algorithm. In particular, it tends to achieve better performance faster, as it is easier to stabilize at good gene values by weighted sum than it is by gene replacing.

Although the improvements of a single percent using random players do seem small, it is a common thing in such a noisy environment as CCGs, and even that leads to a long-term gain. Especially, when it translates into more significant improvement for better player strategies. For comparison, using greedy players for both evolution and evaluation leads to more diversified results, emphasizing the difference between the learning algorithms.

TABLE I

A COMPREHENSIVE COMPARISON OF ALL ALGORITHMS USING RANDOM PLAYER. EACH WAS TRAINED 10 TIMES WITH A COMPUTATIONAL BUDGET OF 1,000,000, YIELDING 50 BEST PLAYERS. EACH TWO PLAYED 20 GAMES ON 500 RANDOM DRAFTS. THE WHOLE EXPERIMENT WAS REPEATED 5 TIMES. ALL SCORES ARE AVERAGED, FOLLOWED BY THEIR STANDARD VARIATIONS. THE BEST RESULTS OF EACH COLUMN ARE IN BOLD.

	$Contest_1$	$Contest_2$	AG_{all}	$AG_{weights}$	AG	Evo_{base}	$Random$	$Random_t$	Average
$Contest_1$	–	48.73 ± 0.60	50.54 ± 0.25	48.35 ± 0.32	49.30 ± 0.29	50.43 ± 0.12	51.38 ± 0.19	51.23 ± 0.15	49.88 ± 0.14
$Contest_2$	51.27 ± 0.60	–	51.95 ± 0.25	**49.67** ± 0.24	50.70 ± 0.28	52.12 ± 0.20	52.87 ± 0.20	52.75 ± 0.20	51.46 ± 0.09
AG_{all}	49.46 ± 0.25	48.04 ± 0.25	–	47.70 ± 0.09	48.49 ± 0.06	49.85 ± 0.08	50.78 ± 0.05	50.59 ± 0.05	49.16 ± 0.06
$AG_{weights}$	**51.64** ± 0.32	**50.32** ± 0.24	**52.29** ± 0.09	–	**50.93** ± 0.07	**52.27** ± 0.03	**53.08** ± 0.05	**52.99** ± 0.04	**51.74** ± 0.04
AG	50.69 ± 0.29	49.29 ± 0.28	51.50 ± 0.06	49.06 ± 0.07	–	51.43 ± 0.07	52.33 ± 0.03	52.11 ± 0.04	50.76 ± 0.05
Evo_{base}	49.56 ± 0.12	47.87 ± 0.20	50.14 ± 0.08	47.72 ± 0.03	48.56 ± 0.06	–	50.93 ± 0.04	50.69 ± 0.03	49.24 ± 0.04
$Random$	48.61 ± 0.19	47.12 ± 0.20	49.22 ± 0.05	46.91 ± 0.05	47.66 ± 0.03	49.06 ± 0.05	–	49.82 ± 0.03	48.26 ± 0.02
$Random_t$	48.76 ± 0.14	47.24 ± 0.20	49.40 ± 0.05	47.00 ± 0.04	47.88 ± 0.04	49.30 ± 0.03	50.17 ± 0.03	–	48.44 ± 0.02

TABLE II

A COMPREHENSIVE COMPARISON OF ALL ALGORITHMS USING GREEDY PLAYER. EACH WAS TRAINED 3 TIMES WITH A COMPUTATIONAL BUDGET OF 1,000,000, YIELDING 50 BEST PLAYERS. EACH TWO PLAYED 20 GAMES ON 500 RANDOM DRAFTS. THE WHOLE EXPERIMENT WAS REPEATED 3 TIMES. ALL SCORES ARE AVERAGED, FOLLOWED BY THEIR STANDARD VARIATIONS. THE BEST RESULTS OF EACH COLUMN ARE IN BOLD.

	$Contest_1$	$Contest_2$	AG_{all}	$AG_{weights}$	AG	Evo_{base}	$Random$	$Random_t$	Average
$Contest_1$	–	51.48 ± 0.44	48.27 ± 0.20	40.75 ± 0.25	42.72 ± 0.35	44.73 ± 0.44	50.49 ± 0.17	51.89 ± 0.67	47.19 ± 0.17
$Contest_2$	48.52 ± 0.44	–	48.81 ± 0.56	42.05 ± 0.39	44.21 ± 0.37	45.94 ± 0.39	52.30 ± 0.16	53.61 ± 0.38	47.92 ± 0.11
AG_{all}	51.73 ± 0.20	51.19 ± 0.56	–	43.38 ± 0.25	45.34 ± 0.03	48.14 ± 0.33	52.77 ± 0.14	54.82 ± 0.17	49.62 ± 0.11
$AG_{weights}$	**59.25** ± 0.25	**57.95** ± 0.39	**56.62** ± 0.25	–	**51.95** ± 0.12	**54.75** ± 0.25	**59.33** ± 0.06	**61.09** ± 0.18	**57.28** ± 0.06
AG	57.28 ± 0.35	55.79 ± 0.37	54.66 ± 0.03	**48.05** ± 0.12	–	52.91 ± 0.12	57.84 ± 0.15	59.58 ± 0.16	55.16 ± 0.12
Evo_{base}	55.27 ± 0.44	54.06 ± 0.39	51.86 ± 0.33	45.25 ± 0.25	47.09 ± 0.12	–	54.96 ± 0.03	56.67 ± 0.09	52.16 ± 0.19
$Random$	49.51 ± 0.17	47.70 ± 0.16	47.23 ± 0.14	40.67 ± 0.06	42.16 ± 0.15	45.04 ± 0.03	–	51.97 ± 0.10	46.32 ± 0.02
$Random_t$	48.11 ± 0.67	46.39 ± 0.38	45.18 ± 0.17	38.91 ± 0.18	40.42 ± 0.16	43.33 ± 0.09	48.03 ± 0.10	–	44.34 ± 0.15

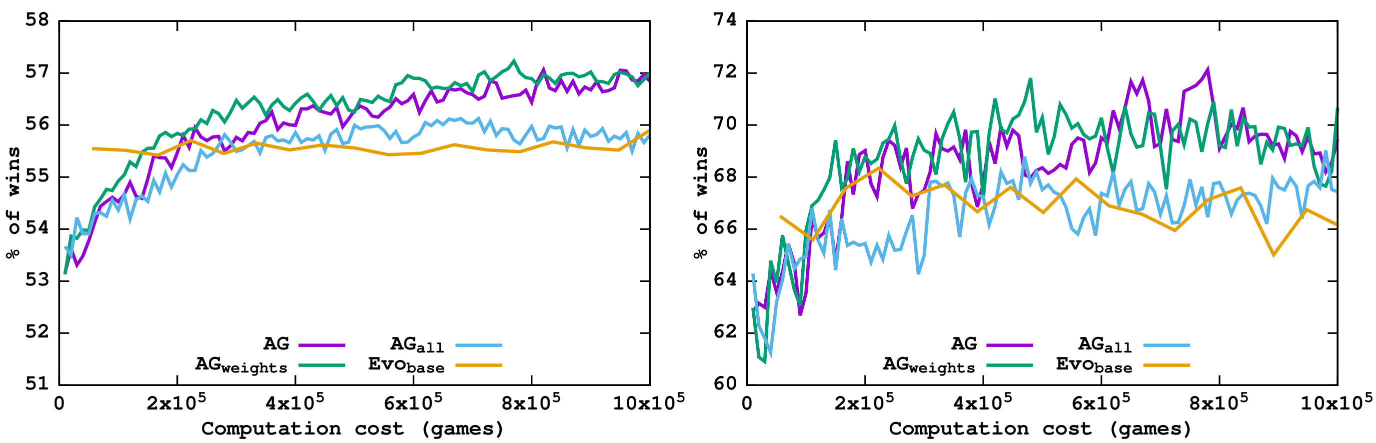

Fig. 3. Process of evolution for AG, AG_{all}, $AG_{weights}$, and Evo_{base} using random (left chart) and greedy (right chart) players. Computation cost (number of simulated games) on x axis, average performance versus $Contest_1$, $Contest_2$, and three random strategies on y axis.

Moreover, a comparison between Tab. I, Tab. II and Fig. 3 shows that even a small advantage over the better performing opponents yields a significant advantage over the less skilled ones.

B. Analysis of $AG_{weights}$ Learning

In Fig. 4, we visualized the performance of overall-best against generation-best individuals during a sample $AG_{weights}$ run to analyze the learning progress. It shows how the all-time top five individuals per 200 generations (the first generation is an entirely random one) perform against the top five individuals of each of the generations on all training drafts.

As we can observe, champions from later generations tend to perform better on average, which is consistent with the previous observations. All tested champions are significantly strong at first because the first generation did not have time to learn. As the learning progresses and the individuals from the following generations are getting better, the scores of the champions tend to be lowering.

It is worth noticing that the best genotypes of each generation were chosen based on the current generation draft, but the chart shows their performance on all d_t drafts. Thus, when we treat champions as constant opponents that differ in strength, we can observe changes in the overall score after learning each new training draft. Each descent means that it improves overall performance, while each rise signals that the overall performance decreased, even though only one particular draft was examined.

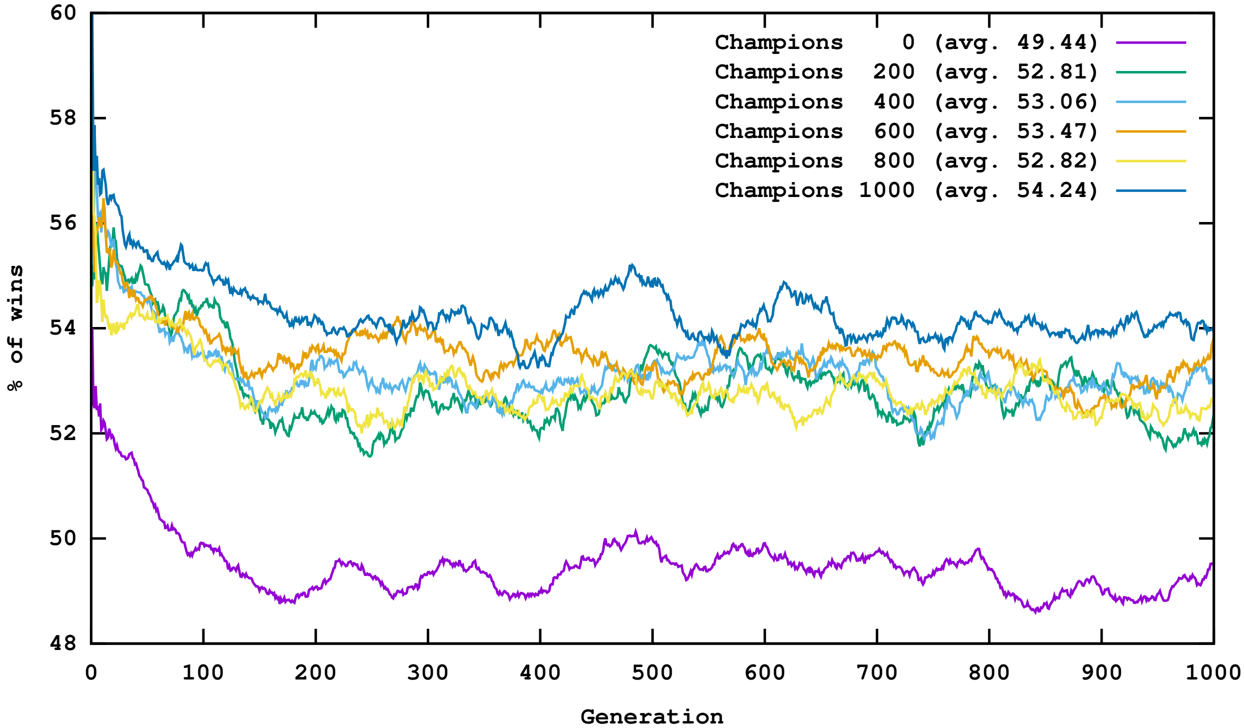

Fig. 4. Average performance of $AG_{weights}$ champions against best five individuals of each generation. Each group played 50 games on every of the 1000 training drafts playing randomly. The average win rate (y axis) tends to drop, as the champions of the following generations (x axis) are getting stronger.

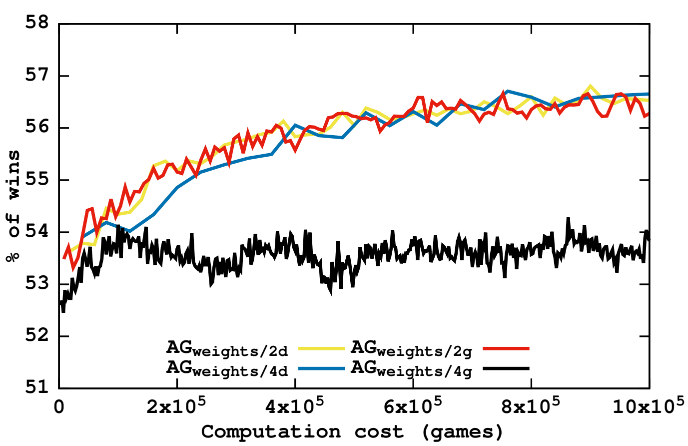

Fig. 5. Process of evolution for $AG_{weights}$ variants using random player. Computation cost (number of simulated games) on x axis, average performance versus $Contest_1$, $Contest_2$, and three random strategies on y axis.

In many places, both peaks and valleys are in similar places for multiple champion lines, which shows the importance of particular drafts.

C. Investigating Active Genes Approach

The performance of active genes evolution correlates with the number of training drafts used to evaluate a single population. More drafts imply that a more significant part of the genotype is active and being affected by the operators. On the other hand, with a limited number of drafts per generation, the quality of the evaluation is limited, and some knowledge may be lost during evolution. Thus, we introduced two additional variants of $AG_{weights}$ algorithm.

The first one, called $AG_{weights/Kd}$, where K is the number of drafts, affects parent tournament and merge phases, in both places adding a loop over the used drafts. To keep the computational cost and the number of drafts the same, this variant runs for g/K generations compared to $AG_{weights}$.

The second variant called $AG_{weights/Kg}$, where K is the number of repetitions, bases on reusing the same drafts K times, leaving the evolution framework as-is. To ensure comparable computation cost, as the variant runs for $g \times K$ generations, the number of games in each generation is accordingly lower.

Analyzing the results from Fig. 5, the average performance of $AG_{weights/4g}$ is the lowest. The budget allowed too few evaluations to make one-generation learning reliable enough. However, less restricted variant $AG_{weights/2g}$, although not the top one, performs reasonably well, achieving higher performance than our Evo_{base} baseline evolution without problems (compare with Fig. 3 for random player).

There is no significant difference between the performance of $AG_{weights/2d}$ and $AG_{weights/4d}$. Both performs similarly, slightly worse than the two leading algorithms. When we compare the difference between those approaches in terms of percent of the genome that is active, we get that it is $\sim$56% for $AG_{weights}$, $\sim$79% for $AG_{weights/2d}$, and $\sim$95% for $AG_{weights/4d}$.

Evolution based on active genes usually performs better when the number of such genes is lower, but this trend is less visible in $AG_{weights}$ variant. In LOCM, proportion of active genes depends on the generation method used to prepare drafts. (This is similar to the *dropout* regularization technique in the artificial neural networks.)

Additionally, we performed two more experiments concerning the trade-off between the number of generations and the number of plays during the evaluation, using larger K values. Both yielded similar but noisier results, therefore have been not included in the paper.

VI. Conclusion

This paper presents initial research towards the problem of deckbuilding in the arena mode of Collectible Card Games. This can be seen as a complementary problem for the standard CCG deckbuilding, where the set of available cards is known in advance. As the domain is characterized by vast state space and omnipresent non-determinism, a straightforward approach to learn draft strategies via an evolutionary algorithm is not very successful and lefts much room for improvements.

In our work, we propose an *active genes approach*, a variant that learns gradually, generation-to-generation. Learning in each generation is based only on the partial training data, and genetic operators are applied selectively only on a subset of genes that is currently considered as relevant. The selection operator is substituted by *merge*, performed between selected pairs consisting of parent and offspring.

We have tested our approach in programming game Legends of Code and Magic, which is used in the Strategy Card Game AI Competition. We designed a few variants of the algorithm and conducted experiments show that usually they perform better than the baseline. Some of them achieve average results that are even significantly better, taking into account that in such a noisy environment as CCG (especially in arena mode), even a small increase in win percentage is a substantial gain leading to overall success in a longer timeline.

What is also important, most of the presented approaches learn very fast in terms of our cost measure (which is the number of required game simulations). Given a fast simulation engine available on the Strategy Card Game AI Competition package [24], it requires about half a minute for a random player to achieve a decent performance on an average run.

For future work, we mainly plan to test active genes family of algorithms on other similar domains, e.g., in Hearthstone. From the game balancing point of view, it would also be interesting to compare policies obtained by one-lane and two-lane versions of LOCM, and how policies trained for one playing algorithm works for the others.

We also aim to develop methods that will improve reliability in terms of achieved performance. This includes using more sophisticated simulated agents (at the expense of computation time) and merging a few runs to perform multiple levels of evolution automatically. We would also like to investigate more algorithm variants and understand what influence on the behavior of the agents has an algorithm that prepares draft choices. Finally, we would like to test how our approach affects agent performance compared to the other solutions presented at the Strategy Card Game AI Competition.

References

[1] M. Campbell, A. J. Hoane, and F. Hsu, "Deep Blue," *Artificial intelligence*, vol. 134, no. 1, pp. 57–83, 2002.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[3] OpenAI, "OpenAI Five," https://blog.openai.com/openai-five/, 2017.

[4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[5] A. K. Hoover, J. Togelius, S. Lee, and F. de Mesentier Silva, "The Many AI Challenges of Hearthstone," *KI-Künstliche Intelligenz*, pp. 1–11, 2019.

[6] A. Dockhorn and S. Mostaghim, "Hearthstone AI Competition," https://dockhorn.antares.uberspace.de/wordpress/, 2018.

[7] Blizzard Entertainment, *Hearthstone*, Blizzard Entertainment, 2004.

[8] J. Kowalski and R. Miernik, "Legends of Code and Magic," http://legendsofcodeandmagic.com, 2018.

[9] A. Janusz, T. Tajmajer, and M. Świechowski, "Helping AI to Play Hearthstone: AAIA'17 Data Mining Challenge," in *2017 Federated Conference on Computer Science and Information Systems*. IEEE, 2017, pp. 121–125.

[10] R. Garfield, *Magic: the Gathering*, Wizards of the Coast, 1993.

[11] Dire Wolf Digital and Sparkypants Studios, *The Elder Scrolls: Legends*, Bethesda Softworks, 2017.

[12] P. García-Sánchez, A. Tonda, G. Squillero, A. Mora, and J. J. Merelo, "Evolutionary deckbuilding in Hearthstone," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, 2016, pp. 1–8.

[13] S. J. Bjørke and K. A. Fludal, "Deckbuilding in magic: The gathering using a genetic algorithm," Master's thesis, NTNU, 2017.

[14] Z. Chen, C. Amato, T.-H. D. Nguyen, S. Cooper, Y. Sun, and M. S. El-Nasr, "Q-deckrec: A fast deck recommendation system for collectible card games," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.

[15] A. Bhatt, S. Lee, F. de Mesentier Silva, C. W. Watson, J. Togelius, and A. K. Hoover, "Exploring the Hearthstone deck space," in *Proceedings of the 13th International Conference on the Foundations of Digital Games*, 2018, pp. 1–10.

[16] V. Volz, G. Rudolph, and B. Naujoks, "Demonstrating the feasibility of automatic game balancing," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2016, pp. 269–276.

[17] F. de Mesentier Silva, R. Canaan, S. Lee, M. C. Fontaine, J. Togelius, and A. K. Hoover, "Evolving the Hearthstone meta," in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–8.

[18] M. C. Fontaine, S. Lee, L. B. Soros, F. De Mesentier Silva, J. Togelius, and A. K. Hoover, "Mapping Hearthstone Deck Spaces Through MAP-elites with Sliding Boundaries," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 161–169.

[19] ADWCTA and MERPS, "Lightforge: Hearthstone Arena Tier List," http://thelightforge.com/TierList, 2016.

[20] HearthArena, "HearthArena: Beyond the Tier List ," https://www.heartharena.com/tierlist, 2017.

[21] CodinGame, "Legends of Code and Magic – Multiplayer Game," https://www.codingame.com/multiplayer/bot-programming/legends-of-code-magic, 2018.

[22] ——, "Legends of Code & Magic (CC05) - Feedback & Strategies," https://www.codingame.com/forum/t/legends-of-code-magic-cc05-feedback-strategies/, 2018.

[23] J. Kowalski and R. Miernik, "Active Genes Evolution for LOCM – source code," https://github.com/acatai/Strategy-Card-Game-AI-Competition/tree/master/paper-activegenes, 2020.

[24] ——, "Strategy Card Game AI Competition – source code," https://github.com/acatai/Strategy-Card-Game-AI-Competition, 2019.