Evolutionary Algorithms for Strategic Card Games

(Algorytmy ewolucyjne w strategicznych grach karcianych)

Radosław Miernik

Praca inżynierska

Promotor: dr Jakub Kowalski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

8 lutego 2019 r.

Abstract

In this work, we deal with the scenario, where the fitness function is non-deterministic, and moreover its value, even for a given problem instance, is impossible to calculate exactly and can only be estimated. Such a situation occurs in popular strategy card games (e.g., Hearthstone, TES: Legends), during the "arena" game mode, where before the match, a player has to construct his deck, choosing his cards one by one from the previously unknown options.

We evolve a card-choice strategy for the programming game Legends of Code and Magic that learns to make a successful choice regardless of the available options, calculating the fitness value is estimated via the simulation-based method.

We investigate the performances of various approaches and 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. Performed experiments show the behavior of this type of algorithms, and points out the variants that tend to learn faster and produce statistically better strategies.

W tej pracy rozpatrujemy przypadek niedeterministycznej funkcji celu, której wartość, nawet w pojedynczym przypadku, jest nieobliczalna i może być tylko szacowana. Takie sytuacje mają miejsce w popularnych strategicznych grach karcianych (np. Hearthstone, TES: Legends) w trybie gry zwanym „areną", gdzie przed meczem gracz musi ułożyć swoją talię, wybierając karty po jednej z nieznanych mu wcześniej propozycji.

Ewoluujemy strategię wyboru kart dla programistycznej gry Legends of Code and Magic w celu wyboru najlepszej, niezależnie od możliwych wyborów, poprzez estymowanie funkcji celu metodą symulacji.

Analizujemy wyniki różnych podejść i proponujemy wariant algorytmu ewolucyjnego, który wykorzystuje koncepcję „aktywnego genu" w celu zmniejszenia zakresu operatorów do zależnych od generacji fragmentów genotypu. Wykonane eksperymenty ukazują zachowanie tego typów algorytmów i wskazują, które warianty uczą się szybciej i prowadzą do statystycznie lepszych strategii.

Contents

Chapter 1

Introduction

Currently, not only classical board games like Chess [1] and Go [2] are used as "grand challenges" for AI research. It has been shown, that modern computer games may also have this role. So far presented approaches to 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 accent is especially put on the certain game features, that makes designing successful AI players especially difficult, e.g. imperfect information, randomness, long term planning, and large action space. One of the game genres containing all these game features is Strategy Card Games, also known as Collectible Card Games.

Recently, numerous research has been conducted in this domain, focusing mainly on MCTS-based approaches to create an agent and creating the deck recommendation systems that will choose the right set of cards to play. The first *Hearthstone AI Competition* [5], with the goal to develop the best agent for the game *Hearthstone* [6] was organized during the IEEE CIG conference in 2018, and the *AAIA'17 Data Mining Challenge: Helping AI to Play Hearthstone* [7] 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 where, unlike the other modes where players can use their full set of cards, players draft a deck from a random selection of cards before every game. Such task is characterized by 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 a variant 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. Individuals forming the next population are no longer selected among the parents/offspring populations, but instead, they are merged from their representatives.

We have conducted a series of experiments to estimate the performance of multiple variants of designed algorithms and compare it to a number of human-made and random baselines. We used the programming game *Legends of Code and Magic* [8], designed especially for handling AI vs. AI games. Algorithms learn on a small sample of random drafts (available card choices) and are tested on the larger number of other drafts. Both processes use game simulations with multiple repetitions to estimate fitness value in such highly non-deterministic environment. The results show that some of the introduced active genes variants perform better than other tested approaches.

Chapter 2

Background

2.1 Strategy Card Games

Strategy Card Game (SCG) is a broad genre of both board and digital games. Starting with *Magic: the Gathering* [9] in early 90s, or more recent *Hearthstone* [6] and *The Elder Scrolls: Legends* [10], 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 in their *hand*, then the actual play begins. A single *turn* consists of a few *moves*, 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 SCG characterizes with a large number of playable cards (over 900 in *TESL* and almost 20,000 in *MtG*), which causes an enormous amount of possible decks compositions. This leads to even bigger in-play search space, as the player does not know the order of their next draws, the content of the opponent's deck nor his in-hand cards. Such numbers tend to increase the importance of 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 "rock-paper-scissors" scheme, but with a larger number of possible types.

2.2 Related Work

Because of the popularity of the game, and available AI simulators, most research has been conducted for Hearthstone, including mentioned before AI competition [5] and data mining challenge [7].

Usually, AI agents are based on the Monte Carlo Tree Search algorithm [11], as it is known to perform well in noisy environments with imperfect information. In some variants [12], which use a pre-analyzed database of decks, and expert knowledge to guide MCTS simulations and heuristic-based state evaluation functions, has been tested against a few chosen meta decks.

The authors of [13] combines MCTS with neural networks and focuses on organizing the search space to reduce its size. Firstly, they propose *chance event bucketing* to ensure uniform sampling across the choices that are significantly different. Secondly, they distinguish high-level actions (cards applied from the player's hand) from the low-level ones (onboard actions) and show that it is possible to achieve good performance by learning the proper behavior only for the high-level ones while using simple policies for the remaining moves.

Another approach that combines MCTS with supervised learning of neural networks has been presented in [14]. This agent uses a novel approach to learn the game state representation based on the word embeddings [15] of the actual card descriptions.

Other attempts rather focus on understanding the game while it is played. Either by predicting the content of the opponent's deck and cards that he is likely to play during a game [16], or by predicting a probability of winning, as in [17]. The latter, authored by the winner of AAIA'17 Data Mining Challenge describes and compares various neural networks architectures that can be used for the task.

The most relevant to our research is a deckbuilding task. So far, all approaches focus on constructing decks statically, from unrestricted sets of cards, and usually test their performance against a small number of predefined human-created decks as in [18]. This work is based on the evolutionary algorithms, which is the leading technique for deckbuilding task, applied not only to Hearthstone, but also Magic: the Gathering [19]. On the other hand, a neural network-based approach to the same task has been presented in [20].

Methods for estimating card values that are useful to arena mode has not been yet subject to proper research, although they are popular among human game players. There exist dedicated web pages and game-helping software that recommends cards [21, 22]. The data they are based on is constantly updated and consists of a mix of expert domain knowledge, mathematical formulas, and remarks made by players on public forums.

2.3 Legends of Code and Magic

Legends of Code and Magic (LOCM) [8] is a small implementation of a Strategy 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.

All cards 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 lines (similarly as in The Elder Scrolls: Legends), so it favors deeper strategic thinking. 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. Because of that, the deckbuilding is dynamic and cannot be simply reduced to using human-created top-meta decks.

LOCM is one of the competitions accepted for the IEEE Congress of Evolutionary Computation 2019, under the name of *Strategy Card Game AI Competition* [23]. So far, the game in a slightly simplified (one lane) form has been used in August 2018 as a CodinGame platform contest, attracting more than 2,000 players (or rather AI programmers) across the world [24]. Fig. 2.1 shows the visualization in the middle of the game.

The big advantage of LOCM for our research is that it is the only game allowing fair AI comparison without much meta influence. As the card choices for the players are different each time, and both players have exactly the same decisions in this phase (which is not true in arena mode in existing computer games, where every created deck is used in several games versus players that had other choices to make).

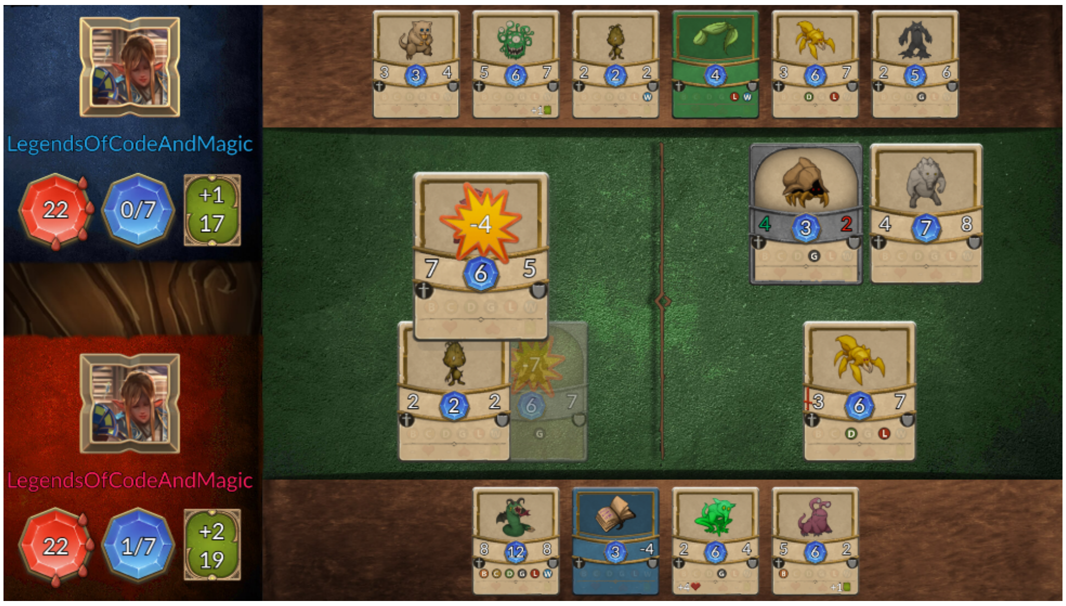

Figure 2.1: Legends of Code and Magic – in-game visualization.

When it comes to the in-game mechanics, LOCM stays in align with the TES: L. There are two types of cards: *creatures* and *items*. Each card has three basic attributes (attack, cost, and defense), three additional attributes, triggered when the card is played, and a set of *keywords*. These additional properties include a bonus card draw and modifications of own and enemy's health.

Keywords are the special card abilities. There is six of them: *breakthrough* (deal excess damage to the opponent), *charge* (summoned creature can attack immediately), *drain* (dealt damage heals the owner), *guard* (must be attacked first), *lethal* (kills any creature it damages), and *ward* (prevents first damage to a creature). All of these take effect during the creature battle.

Creature cards are played on one of the two *lanes*. It stays in this lane, as long as its defense is above zero. Once a turn, each creature with a non-zero attack can battle any enemy creature in the same lane or attack the enemy directly. While battling, both creatures deal the damage at the same time.

Item cards are divided into three subtypes: green, red and blue. Green items can be used on players own creatures, increasing their statistics and/or adding new keywords. Red items can be used on the enemy creatures, reducing their statistics and/or removing keywords. Blue items can be used on both the enemy creatures or the enemy directly.

The game starts with 30 turns of a draft, where both players pick one of three available cards (same options for both players). Then the actual game begins. Both players start with 1 *mana*, used to play the cards. To make it fairer, the second player receives a bonus of 1 mana, as long as he does not use all of his mana in one turn.

Each turn starts with a mana recharge and a card draw (usually one card), up to a maximum of 8 into the *hand*. Next, the player can play his in-hand cards (as long as he has enough mana), attack with his creatures, and finally end his turn. The game ends, when at least one of the player's health is equal to zero.

Chapter 3

Approach

3.1 Problem Specification

Consider two players, A and B. Before every game, they have to choose their decks, i.e., subsets of cards that additionally have to fulfill some game-specific constraints. Then, A and B play against each other, using their playing algorithms and on their chosen decks, randomly shuffled. Thus, the goal of A is to choose deck $\mathcal{D}_A$, and algorithm $\mathcal{A}_A$ such that it performs best over every possible combination of $\mathcal{D}_B$, and $\mathcal{A}_B$. As for existing SCG's the number of possible decks and algorithms are extremely large, research approaches usually focus on one of these tasks and fix the other as a constant factor. Thus, in our case, we assume both players use the same playing algorithm (i.e., $\mathcal{A}_A = \mathcal{A}_B$).

As we mentioned before, so far the research focusing on deckbuilding task was based solely on the scenario where the player (AI) can handcraft any deck from all of the available cards. Thus, given a fixed algorithm, the goal is to generate deck $\mathcal{D}_A$ which has the best win-loss ratio over all possible choices of $\mathcal{D}_B$.

To make this task feasible and more real-life, usually only a few $\mathcal{D}_B$'s are considered, chosen among the strong, human-created decks belonging to a particular meta. The assumption is that the deck good against those representatives will also be good in general (as other decks should be considered worse than those meta decks).

However, considering the arena mode, the task becomes significantly more complicated. Now, the player A is given a set of choices $\mathcal{C}_A$ in the so-called draft phase, and he can choose his deck only from the given (usually very limited) choices. Thus, his goal is to choose the best draft strategy, i.e., a function $\mathcal{C}_A \rightarrow \mathcal{D}_A^{\mathcal{C}_A}$, such that it performs best over every combination of possible B strategies $\mathcal{C}_B \rightarrow \mathcal{D}_B^{\mathcal{C}_B}$.

In existing CCG's arena modes are asymmetric as in the example above, meaning $\mathcal{C}_A$ and $\mathcal{C}_B$ are usually different. However LOCM, to ensure fair AI comparison, provides a fair arena mode, thus for each game, a new set of choices $\mathcal{C}$ is generated, and it is the same for both players A and B.

Fitness Function

This is an example of a problem, for which not only the fitness function is random in a high degree, but even its estimation depends on more random variables.

In such a two-level-deep scenario, a reasonable approach is to sample the space of possible drafts, and estimate fitness based on the performance of draft strategies on those samples. To ensure this approach is sound, we expect that the performance on a low number of training samples is correlated to the performance on a number of unseen test samples. The results presented in Fig. 3.1, confirms such correlation.

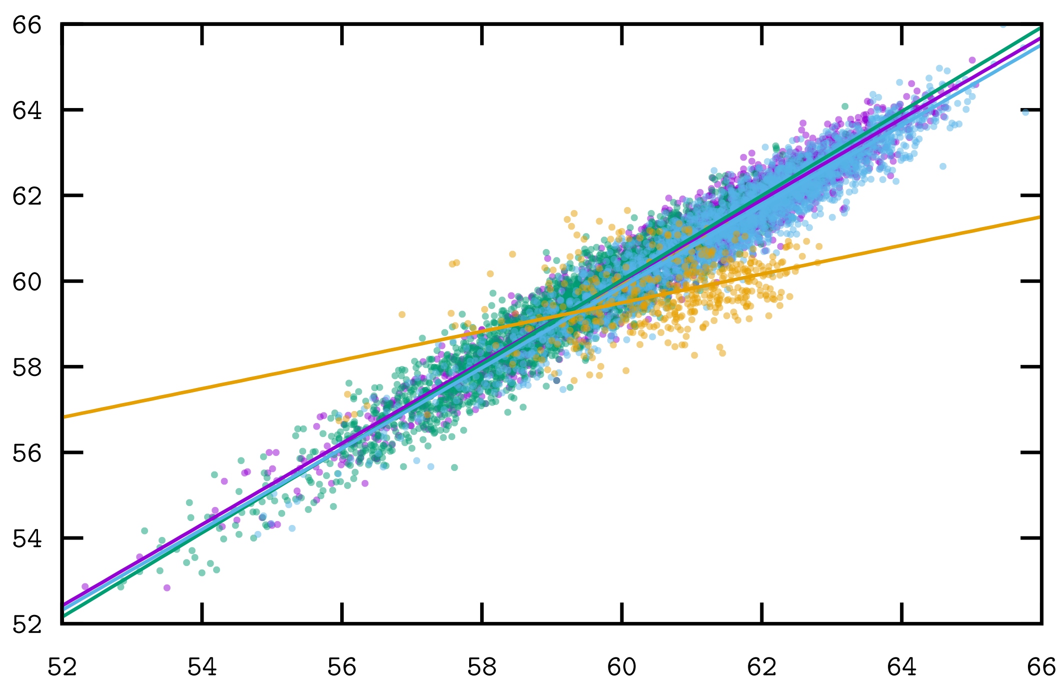

Figure 3.1: Correlation between % of wins on training (x-axis) and fresh (y-axis) drafts for evolutionary algorithms learned on 100 drafts and evaluated on 250 drafts by playing against two human-made, and 3 random draft strategies 250 times.

Domain

Consider a set of all possible cards in the game $\mathbb{C}$. For LOCM, $|\mathbb{C}| = 160$. 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 all draws are independent, the number of possible drafts within the game is $(160^3)^{30} \approx 2.35 \times 10^{198}$.

During the draft, a player knows his previous pickups but is unaware of the future choices. Thus, when learning the best draft function, which is our goal, we search for the best function in the following domain:

$$(\mathbb{C} \cup \{\perp\})^{30} \times \mathbb{C}^3 \to \{1, 2, 3\}, \tag{3.1}$$

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

However, this task can be simplified by discarding the information about previous choices and thus reduces the domain of possible draft strategies to

$$\mathbb{C}^3 \to \{1, 2, 3\}. \tag{3.2}$$

For the practical reasons, it is even more reduced to the pure card-value assignment: in each turn, the card with greater value is chosen. Although taking into account more sophisticated relations, including synergies between cards and controlling a mana curve is no longer possible, this approach is a base for all the real-game helpers (e.g. [21, 22]).

3.2 Base Algorithm

A straightforward evolutionary-based approach to the problem consists of using all available training data in each generation to improve chromosomes that encode priorities of cards.

For genotype representation, we choose a constant-size vector of doubles. Every gene (from 1 to 160) encodes a priority of the associated card. Thus, during the draft phase, the card with the highest priority is chosen in each turn. The population is initialized with the random values between 0.0 and 1.0.

Let n be the size of the population, and d_t be the number of available training drafts. The schema of the base algorithm E_b goes as follows.

To evaluate a population, we play s_g games between every two players for each of d_t drafts (with side change after half of the games). Then, we use tournament selection of size s_{ts} to select $\frac{n}{2}$ parents, based on the number of games won. We use standard uniform crossover and mutation rate m for changing each game into a random new number is 0.05. The elitism size is 2.

Thus, to compute g generations, this algorithm has to play $\mathbb{C}_{E_b}$ games, where

$$\mathbb{C}_{E_b} = n \times (n-1) \times s_g \times d_t \times (1+g). \tag{3.3}$$

3.3 Active Genes Algorithms

Evaluating each generation using all available test drafts, although the most robust, is also time-consuming, and within a constrained computational budget can force evolution to be shallow.

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

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 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 proven itself in the previous generation).

The 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 perform actual games, testing how they behave on the draft that was chosen for this generation. For each parent we perform a tournament of size $s_{pts} = 4$ playing $s_{pg} = 10$ games each round. Selected parents create offspring in the same manner as in the E_b algorithm (uniform crossover, constant mutation rate).

Now, we calculate fitness values for the resulting children population. This is done in s_r rounds, wherein each round we score this 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*. To create each of n individuals belonging to the next population, we use a fitness-based roulette to select a child from the offspring population, and a parent from the parents' population (in this case, it uses the fitness values from the previous generation).

We investigated three variants of the merging procedure. First, called E_a, uses active genes in the most straightforward way. The resulting individual contains values of the active genes copied from the child, and all the other genes are inherited from the parent. Thus, the newest knowledge is considered the most important. Alternatively, E_{aw} is the variant of the E_a, that uses weighted sum for assigning values for the active genes. The final value is 0.75 of the parent gene and 0.25 of the child gene. In this variant, we aim to improve individuals towards good gene values gradually.

For comparison, we also test E_{all} variant, which is based on the same schema but does not use active genes. Instead, the parent is discarded, and only the child gene values are inherited.

Partial pseudocode for these algorithms is presented in Fig. 3.2. Here, we assume that $g = d_t$, which is true for most of the conducted experiments. More sophisticated variants, breaking this assumption, are described separately in Section 4.3. Main procedure EVOLVE proceeds through each generation learning d_t drafts, and calls functions responsible for calculating the children population, and the merge procedure, as described above. Depending on the algorithm variant, the MERGEINDIVIDUALS selects an appropriate merge behavior.

The number of games required to evolve g generations in all these three variants is equal to

$$\mathbb{C}_{E_a} = \mathbb{C}_{E_{aw}} = \mathbb{C}_{E_{all}} = n \times g \times (s_{pts} \times (s_{pts} - 1) \times s_{pg} + s_g \times d_t). \tag{3.4}$$

```
procedure Evolve(generations, options)
    old ← RandomPopulation()
    for generation ← 1, generations do
        draft ← RandomDraft()
        new ← CreateOffspring(draft, old)
        old ← MergePopulations(old, new, draft, options)
    end for
    return old
end procedure

procedure CreateOffspring(draft, old)
    new ← EmptyPopulation()
    for individual ∈ new do
        parents ← SelectParents(draft, old)
        children ← Crossover(parents)
        individual ← Mutate(children)
    end for
    ScorePopulation(draft, new)
    return new
end procedure

procedure MergePopulations(old, new, draft, options)
    merged ← EmptyPopulation()
    for individual ∈ merged do
        a ← Roulette(old)
        b ← Roulette(new)
        individual ← MergeIndividuals(a, b, draft, options)
    end for
    return merged
end procedure

procedure MergeIndividuals(new, old, draft, options)
    merged ← Clone(old)
    if options.mergeAll then
        cardIds ← draft
    else
        cardIds ← allCardIds
    end if
    for cardId ∈ cardIds do
        if options.weighted then
            old[cardId] ← lerp(new[cardId], old[cardId])
        else
            old[cardId] ← new[cardId]
        end if
    end for
    return merged
end procedure
```

Figure 3.2: Active genes algorithm variants pseudocode.

Chapter 4

Experiments

We have tested the performance and analyzed the behavior of the proposed variants of the evolutionary deckbuilding algorithms by playing on a large number of random drafts using the same game tree search method. Algorithms are trained on a set of d_t random training drafts, and their performance is usually evaluated on a set of independently randomized d_e evaluation drafts (see Fig. 3.1 for the justification of correlation). In most cases, a population of size $n = 100$ was tested.

To compare the algorithms in a fair manner, we introduce a cost measure, which is equal to the number of simulated games required to compute the solution, as this is the most computationally significant factor during the evolution process. The computational cost for E_b algorithm is calculated according to formula (3.3), and formula (3.4) describes the cost for the variants with active genes.

To provide a baseline for the quality of evolution, we use two types of randomly generated solutions. First, called R_e creates n random individuals and plays s_g games (with side change) on every training draft between every pair of the individuals. Individuals with the most games won are selected as representatives. Thus the computation cost of this method is:

$$\mathbb{C}_{R_e} = n \times (n-1) \times s_g \times d_t. \tag{4.1}$$

The second baseline, called R_t, randomizes n individuals and performs a tournament to select representatives. To determine the winner of each matchup s_g games on every training draft are played, thus the computation cost is

$$\mathbb{C}_{R_t} = (n-1) \times s_g \times d_t. \tag{4.2}$$

We also use two predetermined, human-made card orderings that originate from the one-lane LOCM contest [25]. Those are called H_1 and H_2 respectively.

All experiments used a random player for playing, to ensure that we evolve a general knowledge about card strength and not their relative strength in terms of given heuristic approach. We have performed a few small-scale experiments with other algorithms, namely greedy and Monte Carlo Tree Search. Both yield similar but less significant results. It might be caused by the quality of used heuristics and the scale itself, and we leave it as a possibility for future research.

In particular, the random strategy leads to a very fast simulation engine. To be exact, it performs around 10000 full games per second on a decent laptop.

4.1 Algorithm Comparison

We compared the overall results obtained by the E_a, E_{all}, E_{aw}, E_b, H_1, H_2, R_e and R_t algorithms. The results for $d_e = 10$, $d_t = 500$, and 5 bests players of 10 runs of each algorithm (up to the cost of 1,000,000) are presented in Table 4.1.

The process of evolution of E_a, E_{all}, E_b, and E_{aw} (and a few more variants we will discuss in the next section) have been visualized on Fig. 4.1. For each of the presented algorithms, best 5 individuals from each generation played 50 games on 250 random drafts against H_1, H_2 and three random individuals. Thus, the results reported on this chart are larger, as the opponents are less skilled.

Scores of the random players are worse than those obtained by evolution, which is a clear indication that learning gives a visible advantage. While the human-authored heuristic H_1 performs gains fewer wins than loses, H_2 seems to be the second strongest choice from this set of strategies. (What is interesting, during the one-lane LOCM contest H_1 was established as the meta, thus outperforming other solutions, including H_2.)

Not surprisingly, E_{all} performs poorly, as its tendency to use only offspring genes result in a forgetting, which is clearly visible when looking at the evolution process. More surprisingly, the performance of the straightforward evolution E_b is also below 50%, and the correlation between performance on the training and evaluation drafts is the worst (yellow line on the Fig. 3.1).

In E_b, generations took significantly longer to compute. Thus 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, yet a few non-exhaustive experiments we additionally performed showed that the score does not rise significantly even with a far greater computational budget. So this variant generates average solutions during the first generation, which he cannot further improve.

On the contrary, the remaining active genes-based approaches E_a and E_{aw} learn step-by-step from a low score, need $\sim$4 times the cost of initial E_b generation to start performing better, and then they continue learning. Which supports that the idea of batched learning and selective genetic operators.

The variant with active genes and weighted merge, E_{aw}, achieve over 1% higher results vs strong opponents, and it seems to behave slightly better vs the weaker ones. In particular, it tends to achieve good performance faster, as it is easier to stabilize good gene values by weighted sum than by gene copying.

Although the improvements of the order of single percents do seem small, it is a common thing in such a noisy environment as SCGs, and even that leads to a long-term gain. Moreover, a comparison between Tab. 4.1 and Fig. 4.1 shows that even the small advantage over high-performance solutions tends to a significant advantage over the less skilled opponents.

Table 4.1: Comprehensive comparison of all algorithms. Each was trained 10 times with computational budget of 1,000,000, yielding 50 best players. Each two played 20 games on 500 random drafts. Whole experiment was repeated 5 times, average scores and standard variations are presented.

	H_1	H_2	R_e	R_t
H_1	–	48.73 ± 0.60	51.38 ± 0.19	51.23 ± 0.15
H_2	51.27 ± 0.60	–	52.87 ± 0.20	52.75 ± 0.20
R_e	48.61 ± 0.19	47.12 ± 0.20	–	49.82 ± 0.03
R_t	48.76 ± 0.14	47.24 ± 0.20	50.17 ± 0.03	–
E_a	50.69 ± 0.29	49.29 ± 0.28	52.33 ± 0.03	52.11 ± 0.04
E_{all}	49.46 ± 0.25	48.04 ± 0.25	50.78 ± 0.05	50.59 ± 0.05
E_{aw}	**51.64** ± 0.32	**50.32** ± 0.24	**53.08** ± 0.05	**52.99** ± 0.04
E_b	49.56 ± 0.12	47.87 ± 0.20	50.93 ± 0.04	50.69 ± 0.03

	E_a	E_{all}	E_{aw}	E_b	Average
H_1	49.30 ± 0.29	50.54 ± 0.25	48.35 ± 0.32	50.43 ± 0.12	49.88 ± 0.14
H_2	50.70 ± 0.28	51.95 ± 0.25	**49.67** ± 0.24	52.12 ± 0.20	51.46 ± 0.09
R_e	47.66 ± 0.03	49.22 ± 0.05	46.91 ± 0.05	49.06 ± 0.05	48.26 ± 0.02
R_t	47.88 ± 0.04	49.40 ± 0.05	47.00 ± 0.04	49.30 ± 0.03	48.44 ± 0.02
E_a	–	51.50 ± 0.06	49.06 ± 0.07	51.43 ± 0.07	50.76 ± 0.05
E_{all}	48.49 ± 0.06	–	47.70 ± 0.09	49.85 ± 0.08	49.16 ± 0.06
E_{aw}	**50.93** ± 0.07	**52.29** ± 0.09	–	**52.27** ± 0.03	**51.74** ± 0.04
E_b	48.56 ± 0.06	50.14 ± 0.08	47.72 ± 0.03	–	49.24 ± 0.04

4.2 Analysis of E_{aw} Learning

In Fig. 4.2 we visualized the relation between the performance of the best individuals against generation-best individuals during the sample E_{aw} run, which allows seeing the processes of "learning" and "forgetting". It shows how the top five individuals per 200 generations (top individuals in generation 0 are actually just random ones) play against top five individuals of each of the generations on all training drafts.

As we can observe, champions from higher generations tend to perform better on average, which is consistent with the previous observations. The initial scores of every champion are high because they play against the early generations which did not have time to learn. As the learning progresses, and the individuals from the following generations are getting stronger, the scores of the champions tend to be lowering.

It is worth to notice that the best genotypes of each generation were chosen based on the current generation draft, but the chart shows their performance on all 1000 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 top individual performance, while each rise signals that the overall performance (although one particular draft was examined) decreased. As we can see in many places, the peaks and valleys are in similar positions for multiple champion lines.

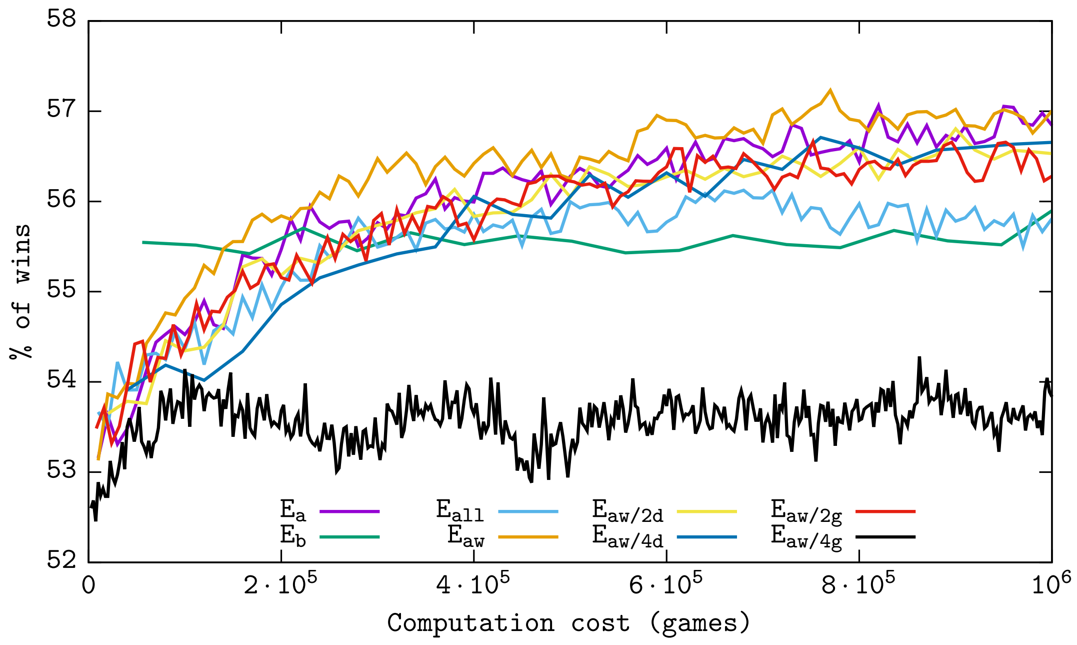

Figure 4.1: Process of evolution for all algorithm variants. Computation budget (correlated with the generation) on x-axis, average performance versus H_1, H_2 and three random strategies on y-axis.

4.3 Investigating Active-genes Approach

Introduction of active genes is correlated with the number of training drafts used to evaluate a single population. More drafts imply a larger part of the genotype is used within the operators. On the other hand, with limited drafts per generation, the quality of the evaluation is limited, and some conclusions may be more easily overwritten during the course of the evolution.

Thus, we introduced two additional variants of E_{aw} algorithm: using increased drafts per generation, and learning on the same draft sequences more than one. The first one, called $E_{aw/Kd}$ where K is the number of active drafts, affects only parent tournament and merge phases, in both places adding a loop over drafts. To keep the number of training drafts and the computational cost the same, this variant runs for $\frac{g}{K}$ generations (comparing to E_{aw}).

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

Analyzing the results from Fig. 4.1, the average performance of $E_{aw/4g}$ is definitely the lowest. Clearly, the budget allowed a too low number of evaluations to make one-generation learning enough reliable. However less restricted variant $E_{aw/2g}$, although not the top one, performs reasonably well, without problems achieving higher performance than our E_b baseline evolution.

There is no significant difference between $E_{aw/2d}$ and $E_{aw/4d}$, both showing similar results and only slightly worse than two leading algorithms E_{aw} and E_a. When we compare the difference between those approaches in terms of percent of the genome that is active, we have that for E_{aw} it is ∼56%, for $E_{aw/2d}$ it is ∼79%, and for $E_{aw/4d}$ it is ∼95%. Which suggests, that the evolution based on active genes performs better when the number of such genes is lower. (This is a similar tendency, as the *dropout* regularization technique in the artificial neural networks.) In the LOCM scenario, this number depends on the generation method used to prepare drafts.

Additionally, we performed two more experiments concerning the trade-off between the number of generations and a number of plays during the evaluation. Both yielded similar but noisier results. Thus, they are not included in the work.

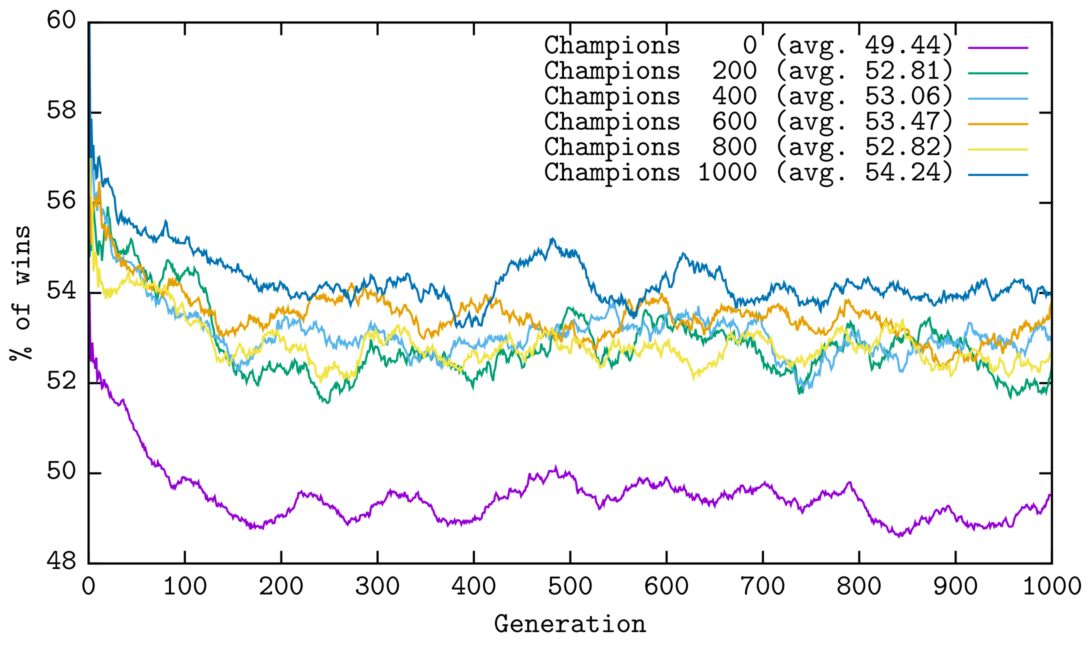

Figure 4.2: Average performance of sampled E_a champions against best 5 players of each generation. Each group played 50 games on every of 1000 training drafts. The average win rate (y-axis) tends to drop, as the champions of the following generations (x-axis) are getting stronger.

Chapter 5

Conclusion

This work presents initial research towards handling the task of deckbuilding in arena mode for Strategy Card Games. This task can be seen as a meta-problem for the standard SCG deckbuilding, where the set of available cards is known in advance. As the domain is characterized by a large state space and omnipresent non-determinism, a straightforward approach to learn draft strategies via an evolutionary algorithm is not very successful.

Instead, we propose a variant that learns gradually, generation-to-generation, based only on the partial training data, but applies genetic operators selectively only on some subsets of genes, and perform merge operation between individuals from the previous population and offspring instead of standard selection to a new population.

We have tested a few versions of this algorithm schema, and observe that usually they perform better than the baseline, and some of them achieve average results that are even significantly better. It is important to notice, that in such a random environment as SCGs, it is unlikely to find a very dominant strategy. Thus even a small increase in win percentage is an important gain that leads to overall success in a timeline of hundreds of games.

What is also important, most of the presented approaches learns 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 [23], it requires about half a minute to achieve a decent performance on an average run.

Bibliography

[1] M. Campbell, A. J. Hoane, and F. Hsu. Deep Blue. *Artificial intelligence*, 134(1):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*, 529:484–503, 2016.

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

[4] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

[5] Alexander Dockhorn and Sanaz Mostaghim. Hearthstone AI Competition. `https://dockhorn.antares.uberspace.de/wordpress/`, 2018.

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

[7] Andrzej Janusz, Tomasz Tajmajer, and Maciej Świechowski. Helping ai to play hearthstone: Aaia'17 data mining challenge. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 121–125. IEEE, 2017.

[8] Jakub Kowalski and Radosław Miernik. Legends of Code and Magic. `https://jakubkowalski.tech/Projects/LOCM/`, 2018.

[9] Richard Garfield. *Magic: the Gathering*. Wizards of the Coast, 1993.

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

[11] C. B. Browne, E Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[12] André Santos, Pedro A Santos, and Francisco S Melo. Monte carlo tree search experiments in hearthstone. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*, pages 272–279. IEEE, 2017.

[13] Shuyi Zhang and Michael Buro. Improving hearthstone ai by learning high-level rollout policies and bucketing chance node events. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*, pages 309–316. IEEE, 2017.

[14] Maciej Świechowski, Tomasz Tajmajer, and Andrzej Janusz. Improving Hearthstone AI by Combining MCTS and Supervised Learning Algorithms. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[16] Elie Bursztein. I am a legend: Hacking hearthstone using statistical learning methods. In *CIG*, pages 1–8, 2016.

[17] Łukasz Grad. Helping ai to play hearthstone using neural networks. In *2017 federated conference on computer science and information systems (FedCSIS)*, pages 131–134. IEEE, 2017.

[18] Pablo García-Sánchez, Alberto Tonda, Giovanni Squillero, Antonio Mora, and Juan J Merelo. Evolutionary deckbuilding in hearthstone. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8, 2016.

[19] Sverre Johann Bjørke and Knut Aron Fludal. Deckbuilding in magic: The gathering using a genetic algorithm. Master's thesis, NTNU, 2017.

[20] Zhengxing Chen, Christopher Amato, Truong-Huy D Nguyen, Seth Cooper, Yizhou Sun, and Magy Seif El-Nasr. Q-deckrec: A fast deck recommendation system for collectible card games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.

[21] ADWCTA and MERPS. Lightforge: Hearthstone Arena Tier List. `https://thelightforge.com/TierList`, 2016.

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

[23] Jakub Kowalski and Radosław Miernik. Strategy Card Game AI Competition – source code. `https://github.com/acatai/Strategy-Card-Game-AI-Competition`, 2018.

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

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