Drafting in Collectible Card Games via Reinforcement Learning

Ronaldo Vieira
Dep. de Ciência da Computação
Univ. Federal de Minas Gerais
Belo Horizonte, Brazil
ronaldo.vieira@dcc.ufmg.br

Anderson Rocha Tavares
Instituto de Informática
Univ. Federal do Rio Grande do Sul
Porto Alegre, Brazil
artavares@inf.ufrgs.br

Luiz Chaimowicz
Dep. de Ciência da Computação
Univ. Federal de Minas Gerais
Belo Horizonte, Brazil
chaimo@dcc.ufmg.br

***Abstract*—Collectible card games are played by tens of millions of players worldwide. Their intricate rules and diverse cards make them much harder than traditional card games. To win, players must be proficient in two interdependent tasks: deck building and battling. In this paper, we present a deep reinforcement learning approach for deck building in arena mode – an understudied game mode present in many collectible card games. In arena, the players build decks immediately before battling by drafting one card at a time from randomly presented candidates. We investigate three variants of the approach and perform experiments on *Legends of Code and Magic*, a collectible card game designed for AI research. Results show that our learned draft strategies outperform those of the best agents of the game. Moreover, a participant of the *Strategy Card Game AI* competition improves from tenth to fourth place when coupled with our best draft agent.**

***Index Terms*—collectible card games, deck building, reinforcement learning**

I. Introduction

Collectible card games (CCG), such as *Magic: the Gathering* and *Hearthstone*, are adversarial turn-taking two-player games that are challenging for humans and artificial intelligence (AI) agents alike [1]. In CCGs, players build decks from a collection of cards and use them to battle against other players. Cards usually represent creatures, items or spells from a fantasy world, and their diverse effects frequently interact, possibly altering the rules of the game in the process. The set of available cards is updated from time to time, making the pursuit for the best deck building and battling strategies a dynamic process. These factors make CCGs much harder than traditional card games such as bridge or poker. Even so, they currently hoard tens of millions of players worldwide between their digital and in paper versions, forming a powerful industry.

Deckbuilding requires strategic reasoning to anticipate upcoming opponents and find effective card interactions, while battling requires reasoning in large state and action spaces to account for the many possibilities of combined card effects. In face of such challenges, the recent advances in game AI that achieved superhuman performance in many games [2]–[4], have not reached CCGs yet.

Game balancing is one of the toughest challenges CCG designers face. Although they conduct careful qualitative playtesting processes before releasing new cards [5], banning freshly-released cards due to them causing unforeseen imbalances in the game is a fairly common event. Fast and strong deck building and battling agents could provide more challenging adversaries for human players and also enable more thorough playtesting [6], [7], helping to detect unbalanced card interactions before they are made available to the public.

In this paper, we present a deep reinforcement learning (DRL) approach to create strong and fast deck building agents for CCGs in the *arena mode*. Differently from most game modes, where a large card pool is available for the players to build decks in an offline manner, arena mode requires the player to build a deck before every set of matches, drafting one card at a time from few randomly presented candidates. Hereafter, we refer to deck building in arena mode as *drafting*.

We model drafting in CCGs as a Markov decision process, and train drafting agents with a DRL algorithm. We evaluate three variants of our approach that differ on how to handle information from previously drafted cards. The performance of the resulting drafters are measured according to the win rate of a battle agent in matches using the respective drafted decks.

We instantiate our approach in *Legends of Code and Magic* (LOCM) [8], a collectible card game designed for game AI research, and reimplement the game engine following the OpenAI Gym [9] interface to facilitate the use of reinforcement learning algorithms on the game. Our resulting draft agents significantly outperformed those of the best LOCM-playing agents. Moreover, we show that a participant of the *Strategy Card Game AI* competition, held at the IEEE CoG 2019 conference, improves from the tenth to the fourth place when using our best drafting agent.

Our main contributions are summarized as follows: (i) a game-agnostic deep reinforcement learning methodology for finding drafting strategies in arena mode of CCGs; (ii) a full-fledged implementation of LOCM's rules as OpenAI Gym environments to train reinforcement learning agents; and (iii) a collection of competitive drafting strategies for LOCM.

II. Related Work

Deck building in all game modes of CCGs is dominated by evolutionary approaches. In those, each individual represents a fixed-length deck, with each of its genes serving as a card slot. Fitness is based on either the win rate [10]–[12] or

the health difference [13] of a fixed battle agent using the respective deck either in an round-robin tournament with the entire population or against a selection of established decks. In a notable variation, the algorithm evolves high-performance decks while also ensuring diversity among them, preventing the common outcome of evolutionary algorithms of converging to a single deck archetype [14].

A machine learning approach has also been proposed, which trains neural networks via reinforcement learning to recommend card replacements in a deck. Competitive decks can then be found by sequentially replacing cards until all original cards have been replaced [15]. A different approach aims to find creative card combos rather than building a full deck [16].

Given a card pool, the aforementioned deck-building approaches search the space of all possible decks looking for well-performing decks. For the arena mode, however, since the space of possible decks is not known beforehand, such approaches are not suitable. Instead, work on arena mode should focus on finding a more general solution, that is, a draft strategy. A single work so far tackles the arena mode. An evolutionary algorithm is used, where each individual represents a ranking of cards instead of a deck. The resulting draft strategy is to choose the highest ranked card on each draft round. The genetic operators are modified to act only on genes whose respective cards appeared in at least one match in the fitness calculation, as these cards are the ones responsible for the resulting fitness [17].

Other unpublished work on arena drafting involve either finding rankings of cards by analysis of thousands of match replays, picking cards to pursue an optimal distribution of resource costs or a combination of both [18]. The Hearth Arena website[1] provides an arena draft helper, guiding human players to draft cards according to card rankings and synergy lists maintained by the Hearthstone community.

We aim to investigate whether deep reinforcement learning algorithms can produce competitive decks, following their previous successes in other domains. We propose approaches that consider synergies with previously drafted cards when picking a new card, which, to the best of our knowledge, no work has explicitly considered so far. Moreover, our approaches operate using card features rather than their IDs, learning what a good card is like instead of which cards are good.

III. Methodology

Our methodology consists in formulating the problem of drafting in CCGs as a game-agnostic Markov decision process (Sect. III-A), and tackling it with deep reinforcement learning, following one of our three approach variants (Sect. III-B).

A. Problem formulation

Let $\mathcal{C}$ be the set of all available cards of the CCG. We consider a generic draft in arena mode, where the player builds its deck incrementally in n turns. At each turn, the player picks one among k cards randomly sampled from $\mathcal{C}$. The player does not know which cards will appear in the future, and, in addition, we assume that the player does not know the cards presented to or picked by their opponents. At the end of n draft rounds, the n-card deck is used in set of battles, which assess its performance.

To apply reinforcement learning, we formulate the drafting problem as a Markov decision process (MDP), whose elements $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ are defined next:

- *The set of states* $\mathcal{S}$ represents all the possible turns in the draft. We define two different forms of state representation. A Markovian state representation, denoted $\mathcal{S}_1$, includes all previously chosen cards by the player as well as the current k cards to choose. Under this representation, the resulting set of states is large: at turn t the agent has already picked $t-1$ cards in the previous turns, yielding $|\mathcal{C}|^{t-1}$ possible combinations, and has $\binom{|\mathcal{C}|}{k}$ possible combinations for the current choice. Considering n turns, the cardinality of $\mathcal{S}_1$ is $\prod_{t=1}^{n}\binom{|\mathcal{C}|}{k}|\mathcal{C}|^{t-1}$. A simpler non-Markovian state representation, denoted $\mathcal{S}_2$, considers only the current card choices, disregarding the past picks. Thus, $|\mathcal{S}_2| = \binom{|\mathcal{C}|}{k}$. This representation yields a much smaller state space at the cost of disregarding possible synergies with previously picked cards. We use these two state representations in our approaches.
- *The set of actions* $\mathcal{A}$ at any turn consists in choosing one of the k presented cards. Formally, $\mathcal{A} = \{1, ..., k\}$.
- *The transition function* $T(s'|s, a)$ is such that, in s', a new sample of k cards is presented if s was not the last draft round (less than n draft rounds have passed). If $\mathcal{S}_1$ is being used, s' also contains the $t-1$ previously picked cards.
- *The reward function* $R(s)$ returns 0 for all non-terminal states and, for terminal states, the result of a battle using the built deck: $+1, 0$ or -1 for victory, draw or loss of the companion battle agent, respectively. We assume a single battle, but a set of battles could also be considered, and the reward would be averaged over these.
- *The discount factor* γ is set to 1 (i.e. no discounting is made), as earlier or later choices of cards have the same influence on winning or losing the battle.

Fig. 1 depicts a sequence of states, actions and rewards of a sample episode in this formulation with $k = 3$ choices per each of $n = 30$ draft turns. Our goal is to find a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ that maps a state to a probability distribution over actions in a way that maximizes the expected reward (i.e. the battle agent's win rate). Our resulting draft agent then uses this policy to pick a card in each draft turn.

B. Proposed draft agents

We tackle the arena drafting MDP with deep reinforcement learning. In each draft turn, the game state is converted to a network-friendly numeric representation that follows either $\mathcal{S}_1$ or $\mathcal{S}_2$ (as defined in Section III-A). This representation contains the relevant features of all current card choices (and previously picked cards for $\mathcal{S}_1$), normalized to the range $[-1, 1]$. Empty card slots (i.e. cards not yet picked) are

[1] https://www.heartharena.com

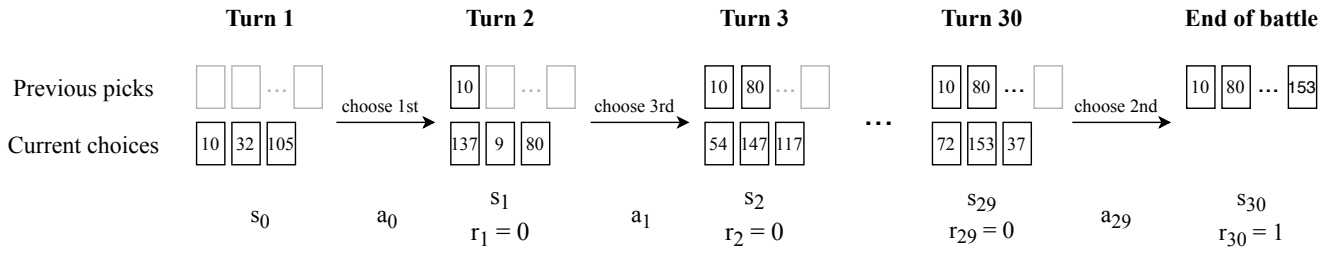

Fig. 1. A sample episode in the MDP that uses the $\mathcal{S}_1$ state representation in a draft phase with $n = 30$ turns. Each numbered rectangle represents a different card in the game. Each state holds all previously picked cards plus $k = 3$ choices of cards. Actions correspond to picking one of the cards. A reward is given at the terminal state representing, in this case, that the companion battle agent has won.

represented by a zero vector. The representation is then given to the network, which outputs the index of its chosen card.

We propose three approach variants that differ in state representation and the type of neural network used by the DRL algorithm. The first variant, **History**, uses $\mathcal{S}_1$ and a multi-layer perceptron (MLP) architecture. As discussed above, $\mathcal{S}_1$ encodes all previously picked cards into the state representation, enabling the agent to consider synergies with them when choosing a next card. The second variant, **LSTM**, uses $\mathcal{S}_2$ but rely on a layer of long short-term memory (LSTM) [19] units to retain information about past picks without explicitly enumerating them. The last variant, **Immediate**, uses $\mathcal{S}_2$ and a MLP architecture, not considering past picks.

IV. Experiments

We present our testbed (Sect. IV-A), training setup (Sect. IV-B), feature extraction process (Sect. IV-C), tuning of hyper-parameters and network architecture (Sect. IV-D), and comparative results of our draft approaches among themselves (Sect. IV-E) and with other drafters in literature (Sect. IV-F). Lastly, we show the improvement of an agent in the Strategy Card Game AI competition when using our best drafter (Sect. IV-G).

A. Testbed

We instantiated our drafting reinforcement learning approach in *Legends of Code and Magic* (LOCM), a two-player digital collectible card game designed for AI research [8]. Its rules are a subset of those of the popular *The Elder Scrolls: Legends*, and deck building is performed exclusively in an arena-like fashion, making it a good testbed for our experiments.

A match of LOCM consists of two phases: draft and battle. In the draft phase, for $n = 30$ turns, players pick one out of $k = 3$ randomly presented cards to form a deck. Both players are presented the same cards but they cannot observe the choices of each other. In the battle phase, players take turns placing creatures in the field, using them to attack the opposing player and their creatures, as well as using item cards, with the objective of reducing their adversary's health points to zero. LOCM is currently used in the *Strategy Card Game AI* competition, hosted by IEEE CoG and IEEE CEC conferences.

To speed-up experiments and facilitate further research on LOCM, we developed an open-source re-implementation of the game engine. We used it to build reinforcement learning environments that follow the OpenAI Gym interface, representing the draft phase, the battle phase and the full game, in one-player and two-player variations. The game engine, environments, experiment code, resulting draft agents and hyper-parameters described in this paper are available on GitHub[2].

B. Setup

In LOCM, the drafted decks are used in a single battle and, since there is a known advantage of playing first in battles [18], the literature leverage this by using separate draft strategies according to whether they will battle as the first or second player. Thus, we trained two separate instances of the deep reinforcement learning algorithm specialized at, respectively, drafting for the first and second players. Each network plays against an earlier version of the other for a determined number of episodes, from which we update both earlier versions with the respective newest version and repeat until the total amount of training episodes is reached. This self-play variant is frequently used when the agents being trained are in asymmetric roles [20].

We trained our drafters partnered with two different battle agents. The first is *max-attack*, one of the baselines in previous LOCM-based AI competitions. It does all actions it can using the cards with the greatest attack power first. The second is a *greedy* agent [17], which picks the best action according to a heuristic state evaluation over resulting states of a one-step lookahead.

We chose the Proximal Policy Optimization (PPO) algorithm [21] to train our drafters, after preliminary experiments with other deep reinforcement learning algorithms of the *stable-baselines* [22] package. PPO uses neural networks to parameterize the policy and an estimate of the value function, which gives the expected future rewards. Each training session consists of 30 thousand self-play episodes of the respective agent, namely History, Immediate or LSTM (see Section III-B), evaluating the drafter in twelve checkpoints during the training session. Each evaluation consists in playing a thousand matches and recording the agent's win rate. To set a common ground between all evaluations, the adversary always

[2]https://github.com/ronaldosvieira/gym-locm

uses *max-attack*'s draft strategy, which chooses the card with greatest attack power.

The amount of episodes of training and evaluation were determined empirically, minding a balance of the training time and the quality of the resulting strategy. On average, a training session lasts 30 minutes with the *max-attack* battler, and 3 hours with the *greedy* battler. Yet, the use of a trained drafter in a draft turn takes no more than one millisecond.

C. Feature extraction

LOCM cards have fifteen attributes. Among these, we select the card type, cost, attack, defense, abilities and health and card draw modifiers, leaving out the card ID number and name, as they do not affect the quality of a card.

With the exception of the card type and abilities, all features are already numeric. We just normalize them by dividing each of these features by their maximum absolute value. To convert the categorical card type, a categorical feature, we apply one-hot encoding. We then convert the resulting binary card type features as well as the six binary ability features (breakthrough, charge, drain, guard, lethal and ward) by considering true and false values as 1 and 0, respectively. This is done on all cards in the game state, and the resulting vectors are concatenated to form the network's input.

D. Hyperparameter tuning

To find the ideal network architecture and hyperparameters for PPO, we used the Tree of Parzen Estimators (TPE) optimization algorithm [23], through the *hyperopt* package[3]. We ran 50 iterations of the TPE algorithm for each of the six combinations between draft approaches (Immediate, History and LSTM) and battle agents (*max-attack* and *greedy*). In each iteration, a training session is conducted using a specific set of hyperparameters. The highest win rate obtained across the twelve checkpoints is returned to TPE, which selects the set of hyperparameters to be tried next, so as to reduce the uncertainty over promising regions of the hyperparameter space.

The hyperparameters tuned for the PPO algorithm were the learning rate, batch size, how many mini-batches are formed from the batch and for how many epochs these mini-batches are used to train, the clip range of the loss function as well as the coefficients of its value function and entropy terms. Moreover, we also optimized the number of hidden layers in the networks (from one to three; in the LSTM approach, the first layer always uses LSTM units), the number of neurons in these layers (from 24 to 256), the activation function (TanH, ReLU or ELU) and the frequency to update the opposing network's parameters in self-play (every 30, 300 or 3000 episodes).

We detected some patterns in the best sets of hyperparameters returned by TPE. As a general trend, larger batch sizes (close to 300) and smaller learning rates (around 10^{-4}) were preferred. Most configurations ended up with shallow 1-layer networks, with the rest settling with 3 layers but less neurons on each layer.

[3]https://github.com/hyperopt/hyperopt

A more evident pattern was that configurations using the *max-attack* battler had an average of 87 neurons in the network, while those using *greedy* had an average of 171 neurons. It may be due to the more elaborate battle strategies of the *greedy* agent, requiring more sophisticated decisions by the networks. Also, the updates of the opponent network were more frequent in *max-attack* configurations (every 336 episodes, on average) than in *greedy* configurations (every 850 episodes, on average). This possibly means that more training episodes are required for the drafter to learn to defeat their opponent during self-play in the latter scenario.

The full sets of hyperparameters found for each configuration can be found at the Appendix.

E. Comparison between draft approaches

In our first experiment, we compared the performance of our three approaches: Immediate, History and LSTM (see Section III-B). Ten training sessions with different random seeds were conducted for each combination of draft approach and battler, using their best network architecture and set of hyperparameters found.

The win rates obtained in each checkpoint were compiled into learning curves, where the drafter's performance and speed of convergence can be observed. Fig. 2 shows the mean learning curves of each approach paired with each battle agent. The first and second player networks are compared separately and averaged at the right-hand side.

The results show a better performance by the Immediate approach in all scenarios, followed by LSTM. Although History achieves better performance than LSTM early in the training, it also seem to settle earlier, while the latter continues to improve. This suggests that, for a training session of 30 thousand episodes, the simplicity of a much smaller state space outperforms the ability of History and LSTM to make decisions considering previously chosen cards. Another hypotheses are that the companion battle agents might not be able to leverage card synergies or that LOCM's cards and rules may be too simple to enable the emergence of relevant synergies or deck archetypes.

Fig. 2 also shows the known advantage of playing first in LOCM (Sect. IV-B), as first players invariably obtained higher win rates than second players. Furthermore, at the end of training, most of the learning curves (especially those of LSTM) still display an increasing trend, meaning that better strategies could probably be achieved by longer training sessions.

F. Comparison with other draft agents

In our second experiment, we evaluated our resulting drafters against draft strategies of different complexities. The first and simplest one is a *random* draft. The second strategy is *max-attack*, which chooses the card with greatest attack power. The last three, named *coac*, *closet-ai* and *icebox*, are draft strategies from the best submissions of past LOCM-based competitions, which draft according to previously calculated card rankings.

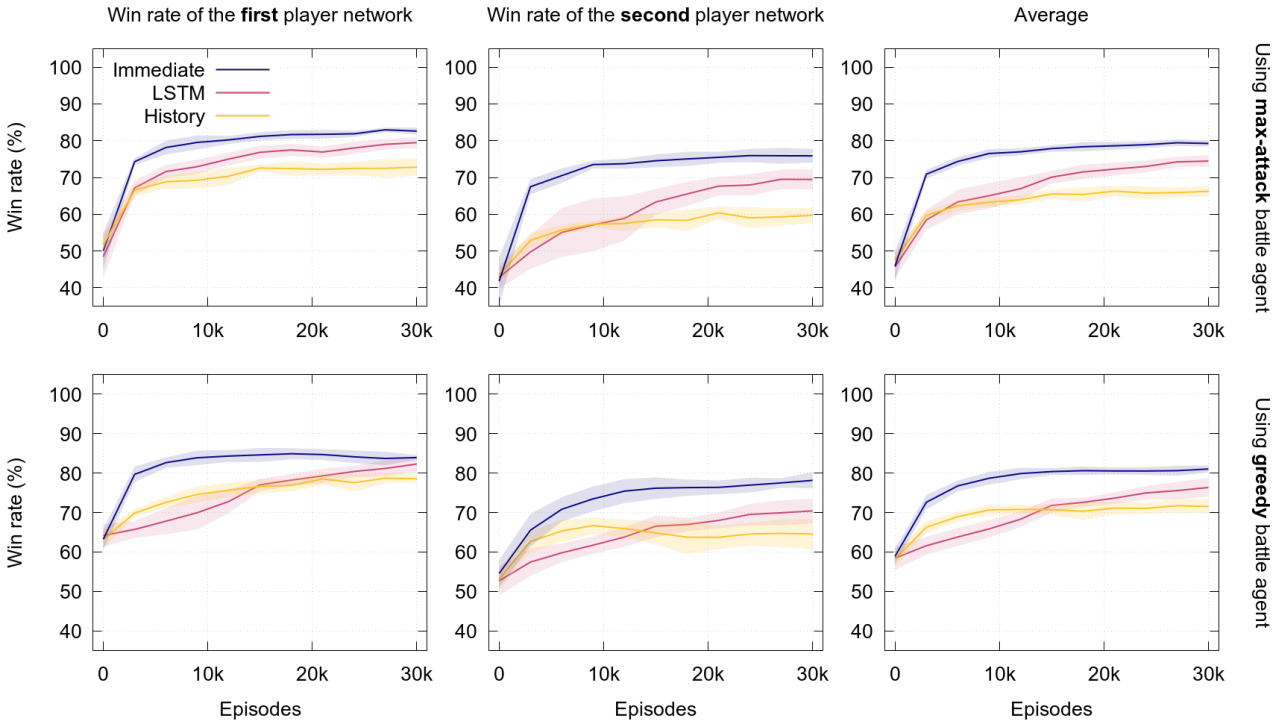

Fig. 2. Learning curves of History, Immediate and LSTM drafters. Left and middle plots show the performance of the networks specialized at drafting for the first and second players, respectively, whereas the right plot shows their average. Plots in the top row are trained and evaluated using the *max-attack* battle strategy, while those in the bottom row use the *greedy* battler. Solid lines and shaded area are the mean and standard deviation of the win rate, respectively.

We measured the win rates of the selected drafters in ten sets of a thousand battles against an adversary that uses the *max-attack* drafter. Since our goal is to find the best performing draft strategies, our agents were represented by the best policy obtained in each of their ten training sessions. Fig. 3 shows the mean win rate and standard deviations of each draft strategy partnered with either *max-attack* or *greedy* battlers.

Our approaches outperform all tested draft agents using either battler. As in the previous experiment, the Immediate approach achieved the best win rates, followed by LSTM and History. We performed paired t-tests to verify the significance of the difference in performance among our three approaches and the best performing competing drafter (*icebox* and *coac*, when using the *max-attack* and *greedy* battlers, respectively). We found all differences to be significant with $p < 0.005$, except the one between the performance of History and *coac* when drafting for second players and partnered with the *greedy* battle agent, with a p-value of 0.076. In that case, we attested a significant difference between the performance of History and the second best performing competing drafter, *icebox*.

Since the draft agents in the literature were created to work with a specific battle agent, the ideal experiment setting would involve training our approaches with each of these battlers in order to compare. However, the computational cost of such task is prohibitive. It is important to mention that current deck-building approaches on the literature also have this issue, and resorted either on experiment settings similar to ours [17] or very limited/no comparative experiments [10], [13], [15].

Nevertheless, when using the *max-attack* battler, our trained networks can win up to 83.5% of the matches on average against the original *max-attack* drafter – evidence that our approaches can find better draft strategies.

G. Agent improvement in the CoG 2019 LOCM tournament

As a comparative way to measure the performance of our best draft strategy, we re-executed the matches of the *Strategy Card Game AI* (SCGAI) competition held at IEEE CoG 2019, modifying the full *max-attack* agent, which was part of the tournament, to use our best draft strategy, namely, Immediate (see Sect. IV-E). The competition gathers the state-of-the-art of LOCM-playing agents in a round-robin tournament, and is, therefore, a good scenario to evaluate our draft strategies.

The source-code of the agents and of the tournament itself are available on GitHub[4]. We used them to run the tournament twice (with the original and enhanced agent) with approximately 3000 matches between every combination of agents, yielding a total amount of approximately 350 thousand matches per tournament. Fig. 4 shows the agents' ranking and win rates in the original and modified competitions.[5]

[4]See https://jakubkowalski.tech/Projects/LOCM/COG19

[5]The results from our reenacted competition are slightly different from the original ones from SCGAI 2019, probably due to hardware characteristics. The most notable difference is the drop in performance of the *Marasbot* agent, that achieved third place in the original competition.

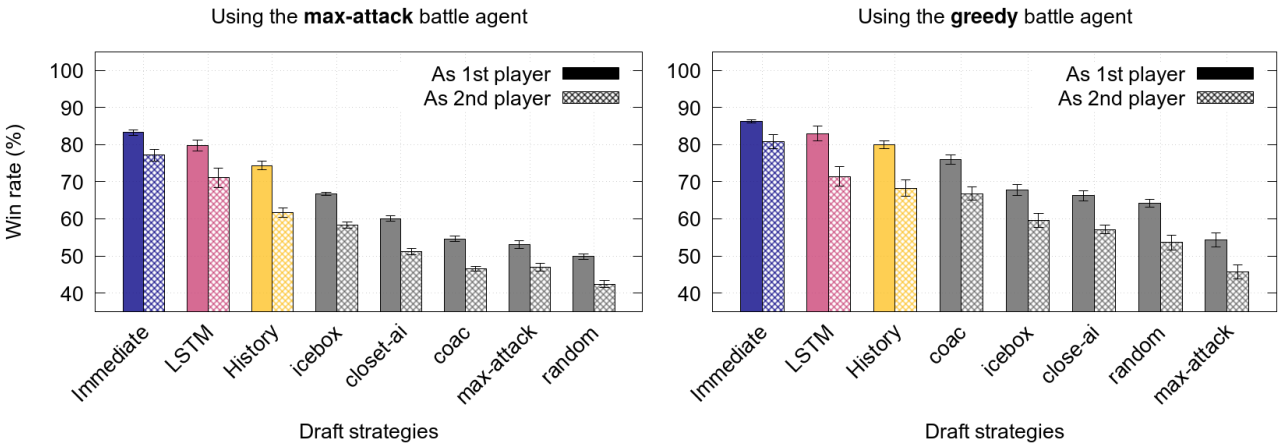

Fig. 3. Performance of the our best drafters plus the current draft strategies in literature, when paired with the *max-attack* (graph in the left) and *greedy* (graph in the right) battle agents. The filled and hatched bars on each agent represent its win rate by drafting for the first and second player, respectively. Error bars show the standard deviations. All win rates were measured by performing ten sets of a thousand battles against a fixed opponent that uses the *max-attack* draft strategy.

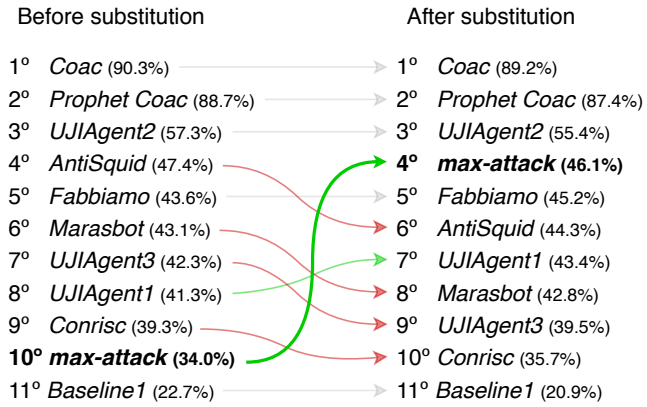

Fig. 4. Tournaments with agents from the IEEE CoG 2019 *Strategy Card Game AI* competition before and after substitution of the *max-attack* agent.

With the aid of our best drafter (Immediate), the *max-attack* agent improved its position from the tenth to the fourth place, winning 46.1% of the matches instead of the previous 34%. Although only trained against a single opponent, our draft strategy was able to increase *max-attack*'s individual win rate against the majority of the opponents in the tournament. In some cases, however, the opponent agents performed better against the modified *max-attack* than against its original version, showing some degree of non-transitivity in the game – what works well against some opponents may be ineffective against others.

V. Discussion

According to our experiments, our reinforcement learning drafting agents, when partnered with the same battle agents they were trained with, outperformed all other selected drafters (Fig. 3). They also improved the ranking of the *max-attack* agent in the reenacted CoG 2019 LOCM tournament (Fig. 4). Our drafting agents learn from their own choices, without domain knowledge or labeled data, and their decision process is fast, as it does not involve lookahead searches.

A direct improvement on our experiment methodology is to use a reward model that reflects the average win rate against a pool of diverse opponents, instead of just one. This might result in more robust drafters, since they will train in a more accurate representation of the scenario they will be used. On the other hand, the computational cost of each training session would increase.

Another hypothesis on generating more flexible drafters is to explicitly account for different playing styles, using ad-hoc teamworking methods [24], where an agent must adapt to previously unseen partners. For example, the drafter could be coupled with different battlers during training, but in addition to the match result, the agent would observe the match execution. This would allow the drafter to infer the playing style of the battler using, for example, player modeling approaches [25]–[27]. The different playing styles of battlers could be mapped to specific deck-building policies.

The surprising result of our experiments is that the Immediate agent, which disregards previously picked cards, achieved the best overall performance. As previously discussed, this suggests that a greater amount of training episodes may be needed for History and LSTM to exploit their ability of considering card synergies. It can also be explained by the fact that our drafters trained with simple battle partners, that may not be advanced enough to exploit combined card effects. The more sophisticated battlers in the literature, that could leverage card combinations, are based on tree-search techniques [28], [29], which unfortunately would slow training considerably.

A possible solution would be to train a reinforcement learning battler as well. With no search involved, it would produce quick responses and enable fast training of draft agents. However, training two interfering policies (draft and battle) simul-

taneously is not trivial: the drafter must learn to generate decks according to an ever-changing playing style, while the battler must learn to play with decks coming from an unstable distribution of decks. Fixing one agent while training the other, and swapping them from time to time could be a possible solution.

VI. Conclusion

This paper tackles deck building in arena mode of collectible card games (CCG), where players draft their deck one card at a time from a set of randomly presented cards. We modeled it as a reinforcement learning problem and presented three approaches to build draft agents that differ on how to consider the previous choices. The performance of a drafter is proportional to the win rate of a battle agent using its decks.

We use Legends of Code and Magic, a CCG designed for AI research, as a testbed to evaluate our methodology. Our draft policies, trained in self-play, outperformed all competing drafters, including the ones used by top agents in LOCM-based competitions, when coupled with two different battle agents. Moreover, our draft improved the ranking of an agent in the reenacted *Strategy Card Game AI* competition.

The discovery of strong drafting policies is a step towards competent AI for CCGs. Once trained, approaches based on reinforcement learning are fast, as they do not need to perform computationally expensive lookahead searches. The benefits of fast and strong CCG players result not only in more challenging opponents for humans, but also contribute to game balancing – one of the toughest challenges CCG designers face – by allowing efficient playtest of new rules or cards. Previous drafting approaches usually build card rankings, requiring retraining to handle new cards. In contrast, our drafters observe card attributes, thus not being tightly tied to the specific card set seen on training. Future studies will aim the construction of a reinforcement learning battler, as well as the use of the presented methodology on more complex commercial CCGs.

References

[1] A. K. Hoover, J. Togelius, S. Lee, and F. de Mesentier Silva, "The many AI challenges of Hearthstone," *KI*, vol. 34, no. 1, 2020.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, and L. Sifre, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, 2017.

[3] N. Brown and T. Sandholm, "Superhuman AI for multiplayer poker," *Science*, 2019.

[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, 2019.

[5] B. Hawley, "Checking in with play design," 2019. [Online]. Available: https://magic.wizards.com/en/articles/archive/play-design/checking-play-design-2019-04-24

[6] F. de Mesentier Silva, S. Lee, J. Togelius, and A. Nealen, "AI-based playtesting of contemporary board games," in *FDG 2017*. ACM, 2017, pp. 13:1–13:10.

[7] N. Tomasev, U. Paquet, D. Hassabis, and V. Kramnik, "Assessing game balance with AlphaZero: Exploring alternative rule sets in chess," *CoRR*, vol. abs/2009.04374, 2020.

[8] J. Kowalski and R. Miernik, "Legends of Code and Magic," https://jakubkowalski.tech/Projects/LOCM, 2018, accessed: 2020-03-27.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *CoRR*, vol. abs/1606.01540, 2016.

[10] P. García-Sánchez, A. P. Tonda, G. Squillero, A. M. García, and J. J. Merelo Guervós, "Evolutionary deckbuilding in hearthstone," in *CIG 2016*. IEEE, 2016.

[11] P. García-Sánchez, A. P. Tonda, A. M. García, G. Squillero, and J. J. Merelo Guervós, "Automated playtesting in collectible card games using evolutionary algorithms: A case study in Hearthstone," *Knowl. Based Syst.*, vol. 153, 2018.

[12] S. J. Bjørke and K. A. Fludal, "Deckbuilding in Magic: the Gathering using a genetic algorithm," Master's thesis, NTNU, 2017.

[13] A. Bhatt, S. Lee, F. de Mesentier Silva, C. W. Watson, J. Togelius, and A. K. Hoover, "Exploring the Hearthstone deck space," in *FDG 2018*. ACM, 2018.

[14] 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 *GECCO 2019*, A. Auger and T. Stützle, Eds. ACM, 2019.

[15] Z. Chen, C. Amato, T. D. Nguyen, S. Cooper, Y. Sun, and M. S. El-Nasr, "Q-DeckRec: A fast deck recommendation system for collectible card games," in *CIG 2018*. IEEE, 2018.

[16] L. F. W. Góes, A. R. Da Silva, J. Saffran, A. Amorim, C. França, T. Zaidan, B. M. Olímpio, L. R. Alves, H. Morais, S. Luana *et al.*, "Honingstone: Building creative combos with honing theory for a digital card game," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 9, no. 2, 2016.

[17] J. Kowalski and R. Miernik, "Evolutionary approach to collectible card game arena deckbuilding using active genes," *arXiv e-prints*, p. arXiv:2001.01326, Jan. 2020.

[18] CodinGame Community, "Legends of Code & Magic (CC05) - Feedback & Strategies," https://www.codingame.com/forum/t/legends-of-code-magic-cc05-feedback-strategies/50996/63, 2018, accessed: 2019-07-14.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, 1997.

[20] B. Baker, I. Kanitscheider, T. M. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[22] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[23] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *ICML 2013*. JMLR.org, 2013.

[24] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein, "Ad hoc autonomous agent teams: Collaboration without pre-coordination," in *AAAI 2020*, 2010.

[25] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron, "Improved opponent modeling in poker," in *ICAI 2000*, 2000.

[26] S. Ganzfried and T. Sandholm, "Game theory-based opponent modeling in large imperfect-information games," in *AAMAS 2011*, 2011.

[27] H. He, J. Boyd-Graber, K. Kwok, and H. Daumé III, "Opponent modeling in deep reinforcement learning," in *ICML 2016*, 2016.

[28] M. Swiechowski, T. Tajmajer, and A. Janusz, "Improving Hearthstone AI by combining MCTS and supervised learning algorithms," in *CIG 2018*, 2018.

[29] D. Wang and T. Moh, "Hearthstone AI: Oops to well played," in *ACMSE 2019*, 2019.

Appendix
Hyperparameters

Table I lists the hyperparameters we optimized, as well as their value ranges and origin. We chose the value ranges empirically and based on previous knowledge about neural networks and the Proximal Policy Optimization algorithm. Tables II and III show the best sets of hyperparameters found to train each approach with the *max-attack* and *greedy* battle agents, respectively.

TABLE I
HYPERPARAMETERS OPTIMIZED IN OUR METHODOLOGY, THEIR VALUE RANGES AND WHERE THEY ORIGINATE. THE AMOUNT OF MINI-BATCHES WERE FIXED AT 1 IN ALL LSTM APPROACHES, DUE TO IMPLEMENTATION CONSTRAINTS.

Hyperparameter	**Value range**	**Origin**
Update frequency of the opponent network	Every 10, 100 or 1000 episodes	Self-play
Depth of the network	1 to 3 layers	Network architecture
Size of the hidden layers	24 to 256 neurons	
Activation function of neurons in hidden layers	TanH, ReLU or ELU	
Batch size	30 to 300 steps	PPO Algorithm
Amount of mini-batches	1 to 300	
Amount of epochs to train with each batch	3 to 20 epochs	
Clipping range of the loss function	0.1, 0.2 or 0.3	
Weight of the value function in the loss function	0.5 or 1.0	
Weight of the entropy term in the loss function	$[0, 0.01]$	
Learning rate	$[1 \times 10^{-2}, 5 \times 10^{-5}]$	

TABLE II
BEST SETS OF HYPERPARAMETERS FOUND FOR EACH OF OUR APPROACHES WHEN PAIRED WITH THE *max-attack* BATTLER.

Hyperparameter	**Immediate**		**History**	**LSTM**	
	1st player	**2nd player**	**Both players**	**1st player**	**2nd player**
Update frequency	Every 10 ep.	Every 10 ep.	Every 100 ep.	Every 100 ep.	Every 100 ep.
Depth of the network	3 layers	1 layer	1 layer	3 layers	1 layer
Size of the hidden layers	52 neurons	81 neurons	93 neurons	25 neurons	24 neurons
Activation function	ELU	TanH	TanH	TanH	TanH
Batch size	300 steps	300 steps	240 steps	240 steps	150 steps
Amount of mini-batches	100	150	120	1	1
Amount of epochs	3 epochs	8 epochs	3 epochs	17 epochs	10 epochs
Clipping range	0.1	0.1	0.3	0.1	0.1
Weight of the value function	0.5	0.5	0.5	1.0	0.5
Weight of the entropy term	8.48×10^{-3}	6.34×10^{-3}	6.59×10^{-3}	2.06×10^{-3}	5.39×10^{-3}
Learning rate	1.38×10^{-4}	1.62×10^{-4}	3.68×10^{-4}	4.57×10^{-4}	4.01×10^{-4}

TABLE III
BEST SETS OF HYPERPARAMETERS FOUND FOR EACH OF OUR APPROACHES WHEN PAIRED WITH THE *greedy* BATTLER.

Hyperparameter	**Immediate**		**History**		**LSTM**
	1st player	**2nd player**	**1st player**	**2nd player**	**Both players**
Update frequency	Every 1000 ep.	Every 1000 ep.	Every 100 ep.	Every 1000 ep.	Every 1000 ep.
Depth of the network	1 layer	1 layer	1 layer	1 layer	3 layers
Size of the hidden layers	169 neurons	154 neurons	199 neurons	199 neurons	51 neurons
Activation function	ELU	TanH	ELU	ELU	TanH
Batch size	270 steps	270 steps	270 steps	270 steps	210 steps
Amount of mini-batches	135	135	135	135	1
Amount of epochs	20 epochs	5 epochs	4 epochs	18 epochs	16 epochs
Clipping range	0.1	0.1	0.3	0.1	0.1
Weight of the value function	1.0	0.5	1.0	1.0	1.0
Weight of the entropy term	5.95×10^{-3}	7.93×10^{-3}	7.23×10^{-3}	4.36×10^{-3}	8.55×10^{-3}
Learning rate	2.28×10^{-4}	1.74×10^{-4}	6.16×10^{-5}	5×10^{-5}	1.11×10^{-4}