Sapienza
Università di Roma

Pausetta Briscola?

A reinforcement learning approach in order to teach an agent how to play the Italian cards game Briscola

Statistical Learning

Students:

Barba Paolo

Candi Matteo

Costantini Silvia

Vestini Maria Vittoria

Professor:

Pierpaolo Brutti

Academic Year 2022/2023

Contents

1 Briscola game

1.1 The rules

Briscola is a popular Italian card game. The deck consists of 40 cards with 4 different suits: swords, coins, cups, and clubs. The cards are the following, in order of importance: Ace (11 points), Three (10 points), King (4 points), Knight (3 points), and Jack (2 points), while 7, 6, 5, 4 and 2 all have 0 points.

At the beginning of the game, each player is given three cards and another card is selected as the Briscola. This last card is known to the players and will be the last one to be drawn in the game. All the cards of the briscola suit are more powerful than the others and will always beat them, regardless of the score ranking. On each hand, A is the first player to play and B is the second one. If A and B play cards of the same suit, then whoever played the card with the higher value wins the hand. If the suits are different and none of them is briscola, A wins the hand. If just one of the two cards is a briscola, then whoever played that card wins the hand. On every hand, the first player is the one who won the previous one.

1.2 Description

The goal of briscola's game is to get more points than the opposite player. There are many possible strategies to achieve more points. Our task is to implement an agent that plays the game with good skills.

Briscola is:

- *Turn-based*: players take turns to make their moves and the game progresses sequentially.
- *Imperfect information game*: each player has partial information about the game itself. Each player knows the cards in their hand, the cards played so far, and the briscola.
- *Stochastic*: Every game starts with a shuffled deck and this introduces randomness in the player's hand cards.
- *Multi-agent*: Briscola can be played with 2 or 4 players. In our case study, we have implemented the game with 2 players.
- *Static*: The environment does not change while a player is thinking.
- *Discrete*: After 20 rounds the game ends.

2 State Space Representation

Briscola game is a challenging task for an agent. There is an inherent uncertainty of states and actions of the other agent, and there is a massive state space.
Briscola is a Partially Observable Markov Game (POMG). It means that agents make decisions and take actions that affect the state of the environment. With some assumptions, such as selecting the opposite player strategy as a greedy one, the game has been transformed into a partially observable Markov decision process. It is composed of uncertain states and well-defined action states that are the cards in your hand.

Game parameters:

- The leading suit (Briscola)
- The cards in the agent's hand
- Agent and opponent's points
- The card in play
- The cards already played

2.1 State Formalization

Any of the 40 cards can take any of these possible states: our agent's hand, the opposite player's hand, already played, the last card of the deck, and in-play card. If a card does not take any of these states, it means that it is in the deck. The agent has access to various information during gameplay, including the cards played so far, the cards in its hand, the Briscola card, the card on the table (if present), as well as its and the opponent's scores. It is evident that the state space is massive and efficient approximation methods are necessary to handle it in practice.

Features	# components
Cards played so far	40
Cards in the hand	40
Briscola card	40
Table card	40
Agent points	1
Opponent points	1
Total	162

3 Agents implemented

3.1 Random Agent

The first baseline agent we implemented is the random agent, which takes a random available action from a given state.

3.2 Greedy Agent

For the greedy agent, we implemented a custom *Min Max Algorithm.* This algorithm assumes that both players play the best possible card, in terms of the points to gain or give to the opponent, each round. The Max player chooses the card that will maximize its win score, while the Min player will always choose the card that will minimize its loss.

Algorithm 1 Min Max Algorithm

```
Input:
  • cards, a list containing the agent's cards.
  • max, a Boolean to indicate whether it needs to maximize or minimize the score.
if $max$ then
    $max\_card \leftarrow$ None
    $max\_value \leftarrow -\infty$
    for $card \in cards$ do
        if $card.value > max\_value$ then
            $max\_value \leftarrow card.value$, $max\_card \leftarrow card$
        end if
    end for
    return $max\_card$
else
    $min\_card \leftarrow$ None
    $min\_value \leftarrow \infty$
    for $card \in cards$ do
        if $card.value < min\_value$ then
            $min\_value \leftarrow card.value$, $min\_card \leftarrow card$
        else if $card.value = min\_value\&\&card.seed = briscola$ then
            $min\_value \leftarrow card.value$, $min\_card \leftarrow card$
        end if
    end for
    return $min\_card$
end if
```

3.3 Q-Agent

In this section, we will explain the structure of the agent implemented via the Q-learning algorithm. Firstly, let us introduce the basic concepts of Reinforcement Learning (RL) and explain the improvements made by Neural Networks. In a RL problem, agents are tasked with executing a sequence of actions to maximize a score. In order to maximize that score, they need to optimize their behaviour. Agents solve RL problems by interacting with their environment and receiving feedback on which actions are good and bad. The environment has been modelled as a Markov-Decision-Process (MDP), a discrete stochastic sequential decision process formalized as a 5-part tuple:
$\langle S,\ A,\ T,\ R,\ \gamma \rangle$ where:

- S is the state space. Set of all the possible states available in Briscola.
- A is the set of all possible actions given a state.
- T: $S \times A \times S \rightarrow [0, 1]$ the transition function, it gives the conditional probability to lead at the state s' taken the action a given the state s.
- R: $S \times A \times S \rightarrow R$ the reward function. The score that the agent is trying to maximize. Is the immediate reward received after transition s' from s.
- γ: the discount factor in order to prevent infinite reward.

Each state in S has state-action pairs called Q-values. The Q-value $Q(s, a)$ describes the rewards an agent can expect from taking action a in state s. Hence, the problem can be reformulated as learning the Q-function.
Due to the massive state space in Briscola, it results impossible to store the q-values. A possible solution is to replace the Q-table with a type of Neural Network called a Deep Q-Network (DQN). Moreover, we chose to train the agent with a fixed number of shuffled decks.

3.3.1 Network Architecture

Replacing the Q-table with a Neural Network introduces some issues while computing the loss. The loss is defined as the following:

$$Loss = |Q^*(s, a) - Q(s, a)|$$

where the Q^* stands for the actual q-value and the Q stands for the outputted q-values from the network.

The optimal Q-value is described with the Bellman Optimality Equation:

$$Q^*(s, a) = E[R_{t+1} + \gamma\ maxQ(s, a)]$$

To estimate the $Q^*(s, a)$ a separate target neural network has been used.

3.3.2 Train

We trained the agent against both a random opponent and a greedy opponent respectively 30% and 70% of the times for a total of 20, 000 Briscola matches. The network is composed of 4 layers RELU activation functions with a decay learning rate. The output of the network is an array of 40 probabilities, obtained with the use of a sigmoid function, associated with the relative's actions.

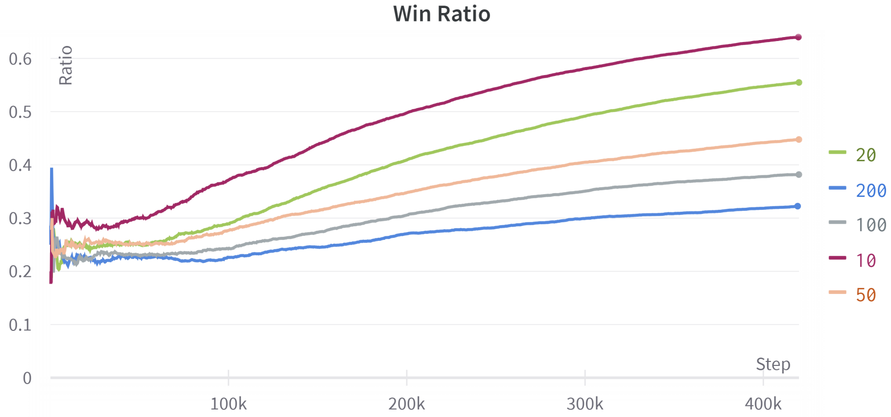

Figure 1: Win Ratio Training

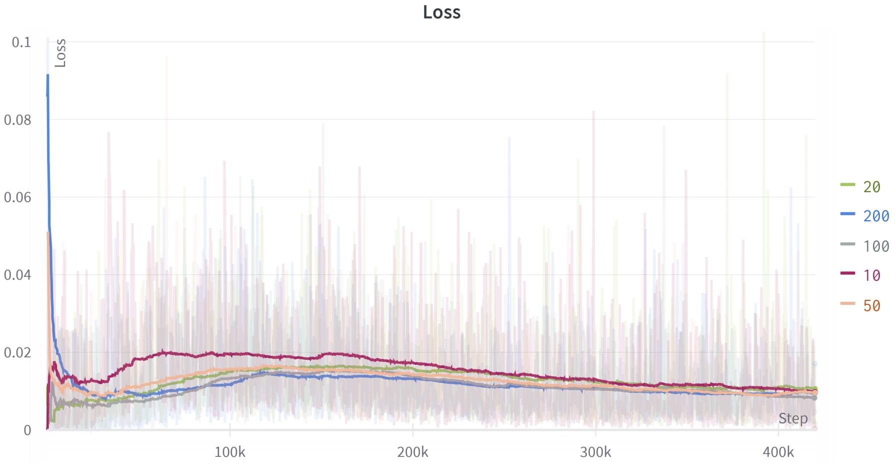

Figure 2: Training L_1 Loss

It has been observed that when the number of shuffled decks is increased, the win ratio after 20, 000 matches decreases. As the complexity of the game increases with the involvement of more decks and matches, it becomes necessary to train the agent more to achieve reasonable performances. Despite this, it is worth noticing that throughout all the training phases, the agents demonstrate a clear improvement in their skills.

3.3.3 Exploration / Exploitation

An important aspect of RL is the exploration-exploitation trade-off. For the agent's training, we have performed an epsilon decay, from a probability $\epsilon = .82$ to perform a random action reaching out $\epsilon = .01$ at the end of the train. This is done since the agent must weigh the benefits of exploring new actions against the benefits of utilizing the best actions it has already discovered.

3.3.4 Invalid action masking

A huge constraint is that the agent must not play cards that are not in their hand. To ensure that, we implemented a mask, giving the invalid actions negative values:

$$\pi(\cdot|s) = mask(softmax(l))$$

where

$$mask(a) = \begin{cases} p_i & \text{if } a_i \text{ is a valid action} \\ -1 & \text{otherwise} \end{cases}$$

- l : the last layer of the network.
- s : the current state.
- a : the set of all actions.

3.3.5 Reward structure

The main goal of a RL agent is to maximise a reward signal. It discovers the actions that lead to the highest reward by trying them, and with its actions, the agent influences both the immediate reward and the successive situations. So, the agent must learn how to balance immediate and future rewards.

In this particular case, the reward structure is a combination of the points earned at each hand and the win or loss of the game, once it is done. For every hand that is **not** the last one of the game, the reward is equal to the gained points normalized by the highest possible score in a hand of Briscola (which is 22). Instead, for the last hand the reward is simply 1 if the agent won the game and 0 otherwise.

4 Results

4.1 Q-Agent vs Random and Greedy Agents

We let the Q-Agent play against both the random and the greedy agent after the training phase. Each time we trained, we tested the model with 1000 matches against the other two agents.

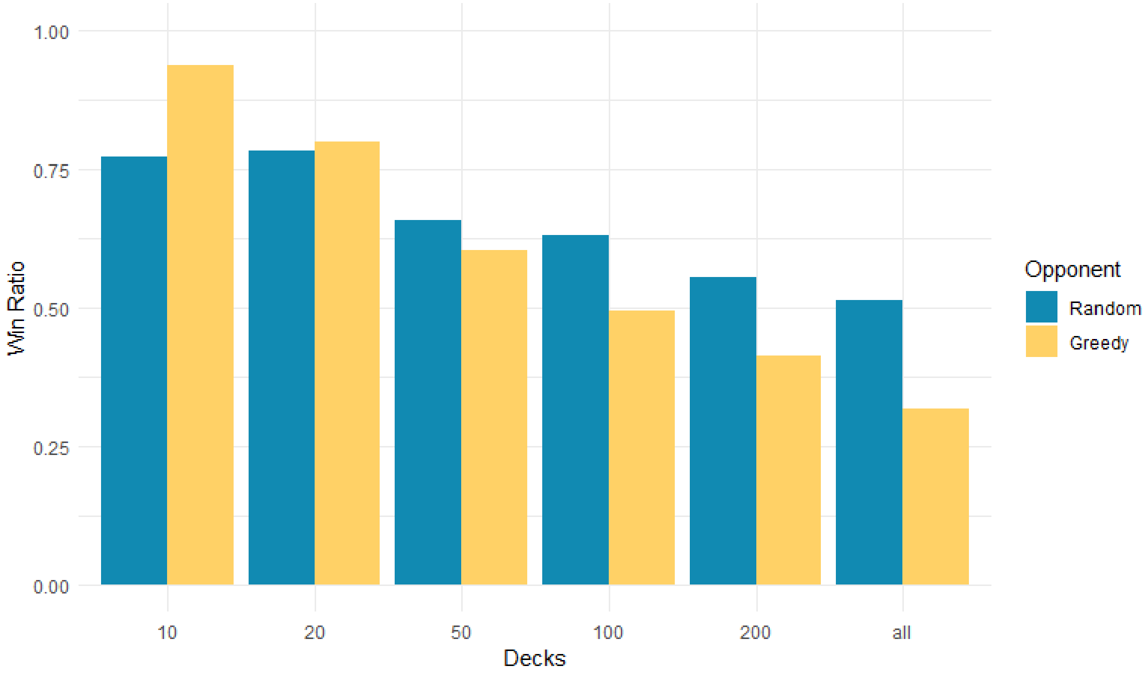

Figure 3: Win ratio Q-Agent vs Random and Greedy agents

The impact of increasing the number of shuffled decks on the model's performance is evident, leading to a decrease in the win ratio. As the number of decks increases, the agent faces a more extensive state space, requiring enhanced learning capabilities.

When employing 20 and 30 decks, the Q-Agent exhibits better performances with the Greedy Agent than against the Random one. However, in the remaining case studies, the Q-Agent demonstrates better performances against the Random Agent.

In the case study involving *all combinations*, the agent's performance aligns with the one of the Random Agent.

4.2 Q-Agent vs Human Agent

Through a Google form, we let people from Data Science course play against our agents through the online version of the game available here: Pausetta Briscola online version. In the questionnaire, we asked the students to rate their Briscola skills on a scale from 1 to 5, the result of the game (won or lost) and their score. We received 23 feedback reports. We are aware that people could play several times and report the match result they prefer. This may lead to a sample bias, anyway this is not the main purpose of our work. The results are shown below. Players who declare a good level at Briscola (greater than 3), on average, beat the agent with a score $76 - 44$. Instead, people with lower skills (≤ 3) score on average 58 points against the 62 of the agent.

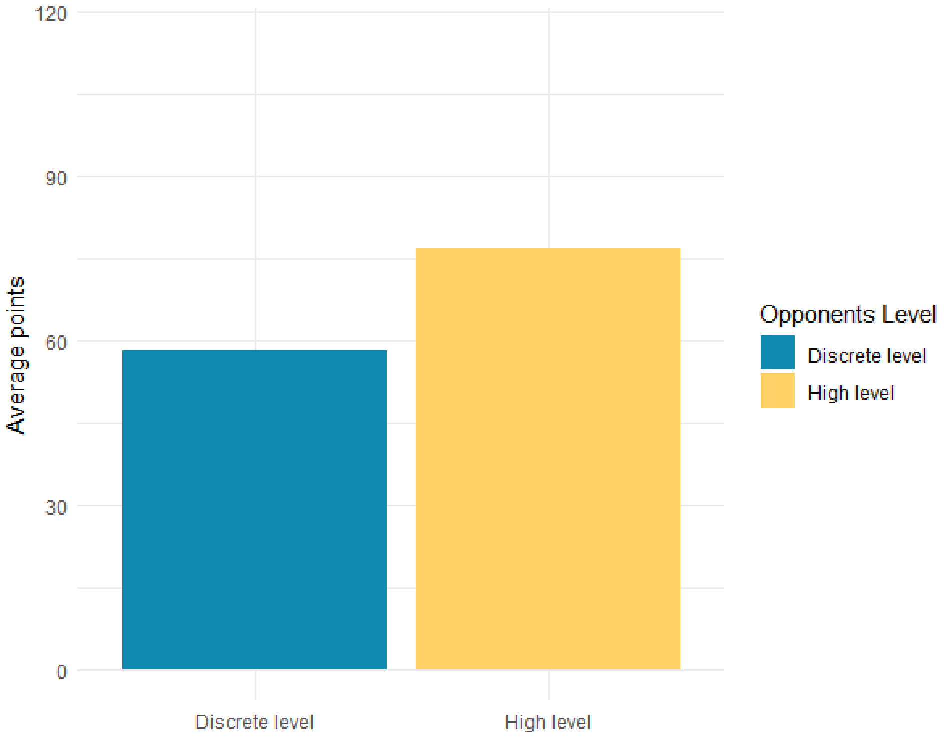

Figure 4: Average points of Human Agents divided per class

5 Comments & Final Remarks

The results of the matches played against the Random and the Greedy algorithms indicate that the agent has successfully learned how to play against them. The results from the matches involving Human players demonstrate that individuals with lower Briscola skills achieve worse outcomes compared to those with more experience. Additionally, it appears that a trained human agent exhibits superior skills compared to the implemented agent.

One potential application for future work is to employ different training approaches for the agent. This could include using other algorithms into the training process or even leveraging the agent to play against itself. Such variations in training methodologies may lead to further improvements in the agent's performance and capabilities.

References

[1] Lettera Unica. Briscola bot. *https://github.com/LetteraUnica/BriscolaBot*, 2022.

[2] Adam Paszke. Reinforcement learning (dqn) tutorial. *Pytorch*, 2023.

[3] Parand Mohri Joaquin Monedero Simon Alexander Safi Alisa Todorova Hemachandra Konduru, Oscar von Moeller. Applications of adversarial search and machine learning in briscola. 2019.

[4] Maxwell De Jong. Lost cities: Using deep q-networks to learn a simple card game. *https://maxwelldejong.com/machine*$_{l}$*earning/lost*$_{c}$*ities/*, 2021.

[5] Wikipedia. Briscola-wikipedia, 2004. [Online; accessed 22-July-2004].