GRENOBLE INP - ENSIMAG

ÉCOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE ET DE MATHÉMATIQUES APPLIQUÉES DE GRENOBLE

THIRD YEAR PROJECT REPORT

(Projet de fin d'études)

PERFORMED AT ISHII LABORATORY

KYOTO UNIVERSITY

Artificial Intelligence methods applied to Contract Bridge (card game)

HUGO HADJUR

3rd year - Specialization MMIS

October 2017 - September 2018

KYOTO UNIVERSITY
606-8501 Kyoto, JAPAN

KYOTO U. Supervisor:
Dr. Shin ISHII

ENSIMAG Tutor:
Prof. Olivier GAUDOIN

Abstract

Artificial intelligence algorithms have been known to perform well at board games (Backgammon, Chess, Go, ...). Their level surpass best human players' when situations are easy to represent, and when observations give 100% of total information. However, contract bridge is a card game which does not give such conditions (hidden cards). Competing at a high level becomes more difficult for classic learning algorithms. Reinforcement learning is a branch of machine learning, which mimics human learning style. We propose some reinforcement learning techniques applied to Contract Bridge. Using Q-learning and SARSA, and adapting theses methods to bridge, learning agents converge to good players in reasonable time.

Keywords: artificial intelligence, machine learning, reinforcement learning, imperfect information, card game, contract bridge

Contents

List of Abbreviations

AI	**A**rtificial **I**ntelligence
RL	**R**einforcement **L**earning
MDPs	**M**arkov **D**ecision **P**rocesse**s**
SARSA	**S**tate–**A**ction–**R**eward–**S**tate–**A**ction (RL method)
DQN	**D**eep **Q**-**N**etwork (RL method)
NS	**N**orth-**S**outh
EW	**E**ast-**W**est
HP	**H**onour **P**oints
NT	**N**o**t**rump
A	**A**ce (Card)
K	**K**ing (Card)
Q	**Q**ueen (Card)
J	**J**ack (Card)

Glossary (Contract Bridge)

NS & EW

Represents the two pairs (teams) of the game of contract bridge as they would appear on a NS-oriented bird-eye view: N at the top, S at the bottom, W on the left and E on the right. NS plays against EW.

Trick Round of play. Step of the game where each player plays one card in turn, clockwise. After each trick, a point is awarded to the winning team.

Hand Set of a player's cards.

Suit Type of cards: spades (♠), hearts (♡), diamonds (♢), and clubs (♣) (listed from strongest to weakest).

Contract The result of the bidding phase. It is defined by a number (between 7 and 13) and a suit, which can also be notrump. After all cards are played, the contract determines which pair scores points, and the amount of points they score.

Notrump Type of contract where no suit is dominant.

Honour points Metric which defines the strength of a hand, based on the number of high cards.
$\text{HP} = 4 * |A| + 3 * |K| + 2 * |Q| + |J|$

Introduction

Ishii Laboratory[1] is one of Kyoto University's Graduate School of Informatics[2] research laboratory. The main goal of this project is to open perspectives about artificial intelligence applied to contract bridge, which is a 4-player (2 vs. 2) card game. The main prerequisites for this task are: a good set of knowledge in programming, learning theory, and the will to perform research.

The schedule was not precisely set at the start of the project: there was a general guideline, a starting point, as well as previous researches performed at Ishii Lab that dissected other board games' artificial intelligence techniques. Those past researches were shared by different students. Usually, the first phase led to assumptions and a semi-advanced learning agent that could compete at a moderate level. The second main step involved more advances methods and a stronger agent. My project belongs to the first step of such research, and is meant to set the bases.

I started without knowledge about the game of contract bridge: in order to develop gaming agents, an important process is to first understand the game and the involved strategies, as well as reading publications about other artificial intelligence research. Similarly, my experience with machine learning algorithms before the start of the project was limited to what I learned in class: reading publications, tutorials, and documentation in general is key to learn how to apply such methods.

[1] `http://ishiilab.jp/kyoto/en/`
[2] `http://www.i.kyoto-u.ac.jp/en/`

Context

Ishii lab is a research laboratory which belongs to the Systems Science department of the Graduate School of Informatics at Kyoto University. There are six main categories among the research themes:

- Computational cognitive psychology
- Prediction-based models of human auditory system
- Statistical learning
- Reinforcement learning
- Neuroinformatics
- Bioinformatics

The project belongs to the reinforcement learning section. Thus, it complements themes like robot behavior learning in open environments, or modeling mechanisms of human behavior.

Ishii lab offers computational resources to researchers. It includes CPUs and GPUs. They are often used to train heavy models, like any kind of complex learning algorithm applied to a substantial set of data. A large part of the research performed at this laboratory involves computer vision problems: specific brain region representation, brain encoding and decoding system, object detection, and image information transfer are some of the examples.

Around 30 people work everyday at Ishii Lab. In the buildings, there are two main rooms where students and researchers perform their own research work. Conference rooms are available for diverse types of talks or meetings. A lot of conferences take place on a weekly basis. For example, a weekly lab seminar is hosted every Thursday, and each member presents the advancement of their research in turns. There are journal clubs too: after the research presentation, someone quickly introduce a scientific paper that they like, in order to create a discussion and widen scientific ideas.

Ishii lab is organized as follows: each student and each researcher has their own research topic. It is supervised by associate professors or professors, so that each research follows the right path. The research topic tackled during my project is being performed within Shin ISHII (professor at the Graduate School of Informatics & lab director) and Ken NAKAE's (project researcher associate) supervision. During the project, Prof. Ishii and Mr. Nakae are here to have discussions, define the guideline, and introduce new methods to me in order to include them in the research.

Overview

3.1 Main Issue

The goal of this research is to analyze reinforcement learning-based methods applied to contract bridge, and their performances.

3.2 Objectives

All along the project, meetings with Prof. Ishii and Mr. Nakae regularly take place in order to adjust the focus based on the advancement and results.

There are a few unavoidable steps that need to be explored:

- First of all, the game of contract bridge needs to be understood well. The rules and the basic playing strategies are important to know.
- State-of-the-art AI methods are essential to move on and set the guideline of the research. In order to gather them, scientific papers about this topic and other relevant themes must be explored.
- Build a clean implementation of the game, anticipating the fact that the learning algorithms return an action (i.e., a card). There must be functions that take those actions as input and update the environment subsequently.
- Start small to go big. Define the range of focus: there is a lot to explore in contract bridge, so concentrating first on some particular situations is key. Having deep understanding of a fraction of the environment will help to widen the results to the entire game.
- Implement, test, and adjust learning algorithms. Based on the results and the information they bring, shift the focus and keep looking deeper at some specific points.

3.3 Contract Bridge

Contract bridge is a 52 cards, 4-player (N, S, E, W) game. It is a cooperative game within a team, and competitive between teams (NS & EW). For each observation of the table, information is incomplete, since most cards are hidden to other players. In order to win, a pair must score 100 points. The game ends when a team reaches 100 points.

3.3.1 Dealing

At the start of a new game, a deck of cards is randomly shuffled, and each player on the table is given 13 cards. To prepare the next step, players analyze their hand.

3.3.2 Bidding for tricks

The research project only focuses on playing, not bidding. However, in order to understand playing, having a solid grasp of the first phase is necessary.

Bidding is the first important phase of the game. A good bidding often implies scoring some points. This step can be compared to an auction for the number of targeted winning tricks. Usually, the pair with the strongest combined hand should win the auction. In other words, they should aim at winning more tricks since their cards are stronger.

During the bidding phase, players call what contract they want in turn (clockwise). The calls can be any contract higher than the current one, or "PASS". When three "PASS" happen in a row, the bidding phase is finished and the contract is the last call. Let us have a look at a time series representation of a bidding example:
N: 1♡ - E: 1♠ - S: 1NT - W: PASS - N: 2♡ - E: PASS - S: PASS - W: PASS
The NS pair won the bidding phase and the contract is 2♡. They will then try to take 2 + 6 = 8 out of 13 tricks during the paying phase.

The difficulty is that every player hides their hand, so each pair must be able to communicate through bidding calls (they cannot use any other mean of communication). In order to do so, bidding standards are translations from bid to the distribution of one's hand. For example, in the "Standard American bidding", which is the most famous standard, the opening call 2NT indicates that the player has a very strong hand. Each pair knows which is the other pair's bidding standard.

Tricks taken	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Clubs (♣)	20 (1♣)	40 (2♣)	60 (3♣)	80 (4♣)	100 (5♣)	120 (6♣)	140 (7♣)
Diamonds (♢)	20 (1♢)	40 (2♢)	60 (3♢)	80 (4♢)	100 (5♢)	120 (6♢)	140 (7♢)
Hearts (♡)	30 (1♡)	60 (2♡)	90 (3♡)	120 (4♡)	150 (5♡)	180 (6♡)	210 (7♡)
Spades (♠)	30 (1♠)	60 (2♠)	90 (3♠)	120 (4♠)	150 (5♠)	180 (6♠)	210 (7♠)
Notrump (NT)	40 (1NT)	70 (2NT)	100 (3NT)	130 (4NT)	160 (5NT)	190 (6NT)	220 (7NT)

TABLE 3.1: Table of all different contracts that can happen, how many points they represent, and their abbreviation.

The higher the score, the higher the difficulty of the contract. Based on this notion, there is an order for bidding: starting with the weakest 1♣, 1♢, 1♡, 1♠, 2♣, 2♢, ..., and ending with the strongest 6NT, 7♣, 7♢, 7♡, 7♠, 7NT. In Table 3.1, this hierarchy goes from the top left, to the bottom right, column by column.

Other examples:

- If NS wins the bidding phase with a "1 Clubs" contract (abbreviated 1♣), it means that they have to take 6 + 1 = 7 tricks out of 13 to score 20 points. This contract is the weakest of all, since the club suit is the weakest one, and taking 7 tricks would mean that EW takes 6 tricks (the smallest winning margin).
- If NS wins the bidding phase with a 4♡ contract, they have to take 6 + 4 = 10 tricks out of 13 to score 120 points. Winning this contract ends the game because the 100 points limit is reached. However, the difficulty of taking 10 tricks is high and such decision involves having good hands.

3.3.3 Playing the hand

Once the bidding phase is done, here are the new pieces of information about the game:

- The contract, which determines the threshold for winning or losing the game, and the dominant suit. The dominant suit involves specific rules for the playing phase.
- The player who called the last bid is called the declarer (North in the first example). His teammate (South), who is called the dummy, puts his cards on the table so that every player can see them. Everybody else keeps their cards hidden. From that point, 13 rounds of a card-strength competition starts. For each trick, the declarer calls the play for himself and the dummy, whereas players of the opposing pair play their own cards. The player on the left of the declarer starts playing for the first trick (East in our example).

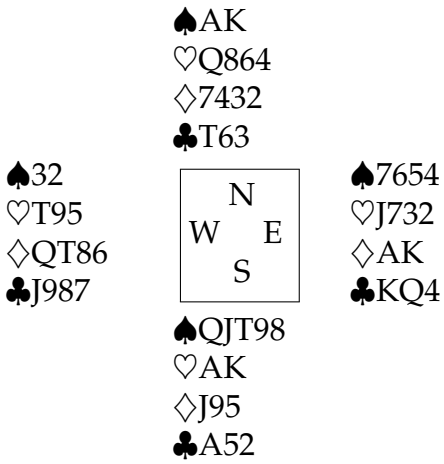


FIGURE 3.1: Bridge hands diagram

Note: for cards of the same suit, the ranking is A-K-Q-J-10-9-8-7-6-5-4-3-2, A being the strongest one.

Notrump Contract

If the suit of the contract is NT, it means that no suit has special powers. To start, each player plays one card in turn (clockwise). The first card on the table determines the dominant suit. If players have cards of the dominant suit, they have to play one of them. Otherwise, they play any other card. If any of the second, third, or fourth card is different from the dominant suit, then it is a losing card. When four cards have been played, the strongest card wins (only taking in account the dominant suit ones).

After the first trick, the four cards are put away, the pair to which the winner belongs leads one trick to zero. Players start the same process again, and the winner

of the previous trick starts (the dominant suit can change at every trick).

Example, if E starts: E: 3♡ - S: 3♠ - W: K♡ - N: 2♡. Here, EW won the trick because K♡ is the strongest card of heart on the table. Based on what S played, it is clear that this player does not have any heart cards left. N played 2♡ because he could not beat K♡, and tried to waste his lowest heart card. W will start to play the next trick.

Trump Contract

If the contract is different from NT, then a suit has special powers. It is the one which appears in the contract name, so player must choose the contract wisely. In trump play, the notion of dominant suit remains, and it is still determined by the first card on the table.

However, the difference is that if a player plays second, third, or fourth, and has no more dominant suit cards, they can win the trick by playing the trump suit. If there are several dominant suit cards on the table, the strongest one wins. If the dominant suit is the trump suit, the trick is played with NT rules, because no other suit overpowers the trump suit.

Example, if the contract is 4♡, and E starts this trick:
E: 3♠ - S: 9♠ - W: J♠ - N: 2♡. Here, NS won the trick because heart suit (2♡) overpowers other suits. Based on what N played, it is clear that this player does not have any spade cards left.

Example, if the contract is 2♢, and E starts this trick:
E: 3♠ - S: 9♠ - W: J♠ - N: 2♠. Here, EW won the trick because J♠ is the strongest card. Every player still had at least one spade card left.

3.3.4 Scoring

At the end of all 13 tricks, each pair counts how many tricks they won and check if the contract have been reached. If it is the case, the declarer's team scores the amount of point that the contract represents, and the other team does not score points. Otherwise, the declarer's team scores zero points and their opponents score some. The higher the number of tricks the declaring side falls short of the contract, the more their opponents score (usually, 50 points per "undertrick").

For example:

- The contract is 4♣, the declarer is S, and the final score is 10 tricks to 3 in favor of NS. So NS scores 80 points and EW scores 0 points.
- The contract is 2NT, the declarer is E, and the final score is 7 tricks to 6 in favor of EW. So EW scores 0 points, and NS scores 50 points.

3.4 Reinforcement Learning

3.4.1 Machine Learning in general

Machine learning regroups all methods which teach machines how to solve problems thanks to examples, and without any explicit logic.

In order to build a machine learning algorithm, there are steps to follow. First, models have to be trained. They must be fed with coherent data in order to learn patterns. Then, performances are tested, whether a test set or new data is involved. It is a general notion, but there are three main types of machine learning algorithms: supervised learning, unsupervised learning, and reinforcement learning.

Supervised Learning

Such techniques involve inputs (also called observations, states, variables) and outputs (actions, classes, vector of probability). Supervised learning gathers techniques which take labeled data as input. The label is the intended outcome. It is similar to learning by seeing a list of examples. For instance, in an image classification problem solved with supervised learning, (`image`, `label`) can be a good form of data for the learning phase. The trained algorithm should then be able to associate the most probable label to input images contains.

Examples of algorithms: Support Vector Machines (SVM), Linear regression, Logistic regression, Decision trees, Linear Discriminant Analysis (LDA), Neural networks, ...

Pros: The focus is easy to specify through the learning data set. Also, in a classification problem, the number of classes is set by the learning inputs (decided by the user).

Cons: If the algorithm has a low convergence rate, it may need to learn from a lot of data. Thus, labeling big data could be a limit.
In the previously introduced example, if the training data set contains images of cats and dogs, the final algorithm cannot recognize cows.
In the case of neural networks, and taking as example image classification, slightly modified images can be misclassified. These are called "adversarial examples" (Goodfellow, Shlens, and Szegedy, 2015). Carefully selected imperceptibly small vector added to the input changes the classification of the image.

Unsupervised Learning

Unsupervised learning is useful when training examples are not labeled. Algorithms take care of finding the right patterns on their own. It is close from human's way of learning: deciding in which situation they are, by observing and finding similarities of things. Such methods are used on selling websites, in order to group items, and build recommendation systems.

Examples of algorithms: K-means, Mixture models, Expectation-Maximization (EM) algorithm, Principal Components Analysis (PCA), Independent Component Analysis (ICA), ...

Pros: It is convenient to gather unlabeled data: it does not require person intervention.

Cons: As the algorithm learns its own way, the results may be hard to explain. In the same way, precision can be low, because the model does not understand the focus. For example, in social networks analysis, the goal can be to analyze users, and cluster them. Hypotheses often describe users groups as: "conversation-driven users", "reserved/silent users", "popular users", ...However, even if humans can naturally pick the right metrics to make such groups, unsupervised learning algorithm are sometimes imprecise.

Reinforcement Learning

Reinforcement learning (Introduced and developed by: Sutton and Barto, 1998) mimics best how humans and animals learn to behave. The important notion is learning thanks to trials and errors, in order to optimize future steps.
The setup is as follows: the agent (program) is progressing inside a dynamic environment. It is given observations, and it has to react with actions. Sometimes, a reward is given, based on how well the agent is doing. Reward appears either after each action, or after a certain time (see Figure 3.2).

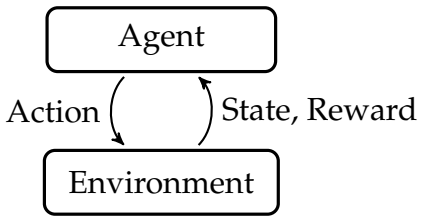


FIGURE 3.2: Reinforcement Learning basic idea. The reward is not always immediate: the result of a behavior can sometimes appear after several actions.

The training phase can be compared to a baby learning how to do things: at first, the agent does not know the right actions to do, so it has to explore a lot in order to converge to a good behavior. It is called exploration phase. The best way to explore is to behave randomly. ε is the exploration rate. In other words, it is the probability of choosing a random action, rather than the theoretical optimal one. During the exploration phase, ε tends to 1.
After it is trained, the RL agent is close to an adult: it acts based on its memory and the rules that it learned in the past. Sometimes, it tries new things, but not as much as before. It is called exploitation phase. The probability of choosing the optimal

move is higher: ε tends to 0.

RL is better than supervised learning at solving problems that generate a high amount of states and actions (i.e. more than a human could handle). In supervised learning, all pairs would need to be labeled in order to train a model. However, in RL, the model trains itself with a trial and error process. In the early 1990s, RL has become popular in the AI and machine learning research field, mostly because that notion. Reward and punishment are assigned, without having to establish how to perform the task.

Markov Decision Processes: There is a link between the setup of a RL problem and MDPs.

Markov Decision Processes	**Example in RL**
Finite state set: S	Every configuration of a contract bridge game.
Finite action set: A	Every possible move.
State transition probability function: $T(s_{t+1}\|s_t, a_t)$	What the opponent does.
Reward function: $R(s_t, a_t, s_{t+1})$	Whether the agent wins the game.
Discount factor: $\gamma \in [0, 1]$	The weight of previous decisions and their outcome.

TABLE 3.2: Examples of MDPs representation in a RL problem (playing bridge).

MDPs also make each state s_t independent from every past states $s_{t-1}, s_{t-2}, \ldots$ In other words, each state alone contains the useful information from the history.

Bellman equation: The Bellman equation is an important concept for solving optimization problems. It uses dynamic programming idea to find the optimal score value V, and is given by:

$$V(s_t) = \max_a \left[R(s_t, a) + \gamma \sum_{s_{t+1} \in S} T(s_{t+1}|s_t, a_t) \times V(s_{t+1}) \right] \tag{3.1}$$

It represents a relationship between the value of a state, the immediate reward, and the value for the future steps. The probability function passes the knowledge of every possibilities s_{t+1} that the environment can give after performing an action a. The Bellman equation averages each of these outcomes and weights them by probability of occurrence. In other words, the value of the first state is equal to the discounted (thanks to γ) value of the expected next state, added to the reward expected along the way.
$V(s_t)$ is the unique solution for the Bellman equation. This equation is the foundation of many optimization problems, including the reinforcement learning ones that are going to be introduced.

3.4.2 Definition of Q-learning

Perfecting games has always been an interest for humans. Along history, researches and computational power have evolved in cycles: computational power helps researchers to test their assumptions, and when methods become too computationally costly, developing more advanced methods becomes the focus while waiting for new machines.

Q-learning is first introduced in order to counter the lack of computational power towards neural networks (Watkins, 1989; Watkins and Dayan, 1992). For a particular state, an action is selected by the Q-learning agent, and evaluated based on its immediate reward or penalty. Generalizing this task to all actions for all states, the best overall moves are determined by long-term discounted reward. It is a primitive form of learning, but it can operate as the basis of more sophisticated devices.

The knowledge of the learning agent is stored in a q-table, which is a 2-dimension table (one for the states, one for the actions). Each q-value $Q(s_t, a_t)$ corresponds to the estimated score when choosing the action a_t at state s_t. It is defined as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{3.2}$$

Where α is the learning rate (how much the current value is updated), r_{t+1} the reward given after performing a_t at s_t, γ the discount factor (the importance of future rewards), and $\max_a Q(s_{t+1}, a)$ the estimated optimal future q-value.

Q-learning updates q-values by supposing that the next action is the best one. It is called "off-policy" strategy.

3.4.3 Definition of SARSA

SARSA is a Q-learning variant. The difference is in the q-value update formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{3.3}$$

Instead of supposing that the next chosen action is optimal, SARSA takes as input the action taken at the next time step, whether it is the optimal one or a random one (decided by ε). This strategy is known as "on-policy".

3.5 Related Work: AI Applied to Games

3.5.1 The evolution of AI for solving games

Even if the rules of contract bridge (Section 3.3) do not directly show how algorithms struggle at mastering it, this game is a true challenge for AI for now and the near future (Ventos and Teytaud, 2017). In order to understand what are the concerns about mastering this game, other games' AI researches and their characteristics need to be developed.

Board games master players are starting to become less and less efficient against computers since about 30 years ago. In 1979, *BKG 9.8* (Berliner, 1980), which is a Backgamon[1] algorithm, beats a world champion for the first time in the history of challenging board games. However, this one game played between the world champion at that time and *BKG 9.8* had some bias because of the rolls of dices. The researcher behind this algorithm admitted that the rolls were strongly in favor of the AI, discrediting the superiority of the program.

Game	AI performances in 1998	AI performances in 2018
Connect Four	Solved	Solved
Checkers	Better than any human (Solved)	Better than any human (Solved)
Backgammon	Better than any human except about 10	Probably better than any human
Chess	Better than all humans except about 250	Better than any human
Go	Worst than best 9-year-old player	Better than any human
Bridge	Worst than best players at local clubs	Well below world class level

TABLE 3.3: Performances of computer in board games in 1998 and 2018
(inspired by Smith, Nau, and Throop, 1998).
Updated for Go (Silver et al., 2017) and Bridge (Ventos et al., 2017).

The first official world championship title won by a program occurs for the game of Checkers[2] in 1994 with the *Chinook* algorithm. The method relies on the fact that there is a proven winning strategy. *Chinook* is a program translation of the perfect steps which lead to a sure win. If both players use it, it is proven that the game results in a draw (Schaeffer et al., 2007).

Checkers is the most challenging solved game, unlike Chess, Go, and Bridge (putting Backgammon aside because of the constant added randomness). These games require solving techniques that are more advanced than heuristic rules. In the case of Chess, IBM's *DeepBlue*[3] beat Garry Kasparov, the reigning world champion in 1997. The strength of this computer is the mix between brute-force tree searching and its computational power, with 200 million move estimations per second.
Recent chess programs outperform humans by an even higher margin: 3200 Elo rating[4] for the best AI, and around 2800 Elo rating for the best grandmasters[5].

New techniques for AI methods needed to be developed, especially for Go, because the complexity of this game is much higher than the ones stated before: the number of possible moves and the difficulty for estimating the quality of a configuration make classic search trees useless. The most famous Go AI, Google's *AlphaGo*[6], is a combination of different AI techniques: deep neural networks, reinforcement learning, and advanced Monte-Carlo tree search. The task is divided into two main parts:

[1] Rules of Backgammon: `http://www.bkgm.com/rules.html`

[2] Other names for Checkers: Draughts (British English), "Dames" (French): `https://en.wikipedia.org/wiki/Draughts`

[3] Characteristics of *DeepBlue*: `http://www.ibm.com/ibm/history/deepblue`

[4] Ranking system: `https://www.thechesspiece.com/what_is_an_elo_rating.asp`

[5] Highest Chess rank: `https://2700chess.com/`

[6] Characteristics of *AlphaGo*: `https://deepmind.com/research/alphago/`

selecting the next move, and predicting the winner. Each of those is performed by one neural network.
Another key point is the environment: who is playing against the program in the learning phase. First, it is trained against high-level human players. Then, it uses reinforcement learning to play against other versions of itself thousands of times and learn from its errors.
After *AlphaGo*'s victory over world champion Lee Sedol, Google's DeepMind laboratory kept improving the performances and released *AlphaGo Zero* in late 2017 (Silver et al., 2017). This version does not use any human input, and is entirely self-taught. Also, the structure of neural networks and their inputs are modified.

3.5.2 Imperfect information games AI research

Mastering Go is the latest hot topic in game AI. However, many wonder what could be the next step. How to widen the capabilities of computer in games? As a reminder, all games described so far are perfect information games: everyone sees everything that is happening at all time. Some recent reports state that the new challenge could be imperfect information games. Card games like contract bridge, hearts, poker or spades remain hard to master for machines, mostly because of the lack of information and the psychological aspects (cooperation, bluff) of these games.

Partially Observable MDPs (POMDPs): POMDPs is a variant of MDPs, where each observation does not express the full environment. It introduces estimation functions which aim at finding which real state is behing observations.

In POMDPs acting and planning methods (Kaelbling, Littman, and Cassandra, 1998), an internal belief state is added to the agent and summarizes its previous experience. The state estimator updates the belief state thanks to the last action, the current observation, and the previous belief state. The policy generated actions based on the belief state.
The belief state is a probability distribution over all possible states.

Card games researches: Hearts is an imperfect information card game that has been studied with POMDPs (Ishii et al., 2005; Fujita and Ishii, 2007). Each state is estimated using the history of the current game, and actions are selected with RL learning methods (Temporal-Differences learning).
Deep RL is an efficient tool to get some results, and was already applied to poker (Heinrich and Silver, 2016).

Bridge AI has also been explored. Of course, the two main different phases of the game (bidding and playing) add difficulty. Usually, researches only focus on one of them. An important bridge technique is double-dummy analysis (Berlekamp, 1963): supposing that every hidden card in the game and the contract are known, there is an optimal way to play for each player, and a unique optimal score.

Yeh and Lin, 2016 use deep RL in order to predict the distribution of suits in players hands, using the sequence of bidding calls. Thus, they introduce a new bidding standard which provide more information than the American Standard.
WBridge5 (Ventos et al., 2017) is the current world computer-bridge champion. It uses boosting and seeding, and relies on double-dummy analysis. Seeding involves generating samples of hidden card distributions based on the context. Such hypotheses are used in order to make the best move. This program cannot beat the best humans yet.

In this project, such belief representation is not directly modeled. The target is to teach a RL agent how to indirectly estimate it, thanks to direct observations of the game.

Methods

4.1 Contract Bridge Game Implementation

The very first concrete implementation step is meant to build an environment for the learning agent. A RL environment needs to have rules which create new state based on an action applied the previous one. It represents the state transition function of an MDP.
In other words, contract bridge must be implemented such that given a state s_t at time t and an action a_t, the RL agent should receive an updated state s_{t+1} based on how the opponents play.

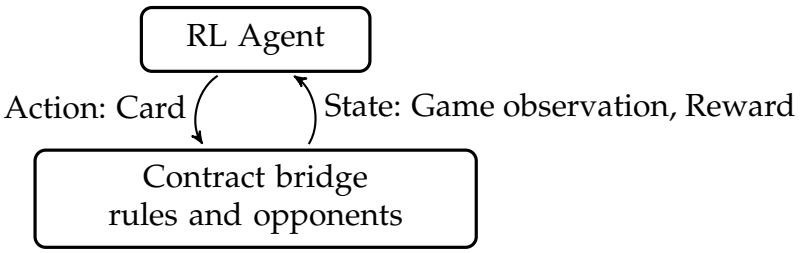


FIGURE 4.1: Contract bridge RL representation. Reward is given at the end of each trick.

In order to apply RL methods, some key-points of the game implementation must be considered:

- **Class representation and major functions**: Object-oriented implementation in Python3 was used. `Suit`, `Card`, and `Player` classes are the bases of the implementation. The main class is `Game`, and each instance corresponds to a specific game of contract bridge. A game includes four `Player`, which all have a different `Position` and a list of `Card`, as well as the contract information, whose turn it is to play, the current trick (hashtable with `Postion` as key, and `Card` as values), and the memory of the game (all previous tricks).

  Such implementation enables to have access to all game's variables, and to return the one that the RL algorithm needs.

- **Transition function implementation**: This function takes a card as parameter and update every variable thanks to the rules. It updates the winner if it changes. It adds the trick to the history, and resets it if the current trick is done (if a multiple of 4 cards has been played in total).

- **Opponents' strategy**: Our RL methods focus on teaching an agent how to play against a fixed opponent (not another agent). Different implementations are made to compete against the agent.

  First, opponents play their cards randomly. It is the easiest implementation, but also the one which give the least difficulty. Thus, the agent is not prepared to play against good opponents, and may struggle to perform at high level.

  In order to theoretically strengthen the AI, a better implementation is made from a rule based agent. It consists of a sequence of manually generated conditions, aiming at making the best possible move in specific situations. For example, if an opponent plays last and has a chance to win a trick, then he plays the right winning card. In theory, the more advanced opponent, the better the agent.

- **Observation**: A function which returns the observation based on the position of a player is implemented. For example, in a real life game, the declarer sees at all time his and the dummy's hands, and the current trick on the table. He also tries to remember the past tricks, in order to estimate the opponents' hands. In the implementation, this comes as a list:
  [`declarer's hand, dummy's hand, current trick, trick history`].

4.2 Complexity vs. Information Tradeoff

4.2.1 Complexity Problem

As stated in Section 4.1, there are a few variables that can be observed at all time. After each move, they evolve, creating a new observation. The important point here is to keep in mind how many observations there are. In the case of the declarer's point of view, even if opponents' cards are hidden, each distribution of cards is almost unique. After the i^{th} trick, the observation is:

- `Declarer's hand`: 13 - i cards.
- `Dummy's hand`: 13 - i cards, which represents $\frac{(52-2i)!}{26!}$ possibilities when combined with the declarer's hand.
- `Current trick`: 1 card taken from each player's hand, i.e. $(13-i+1)^4$ outcomes.
- `Trick history`: All the previous tricks, i.e. $\left(\frac{13!}{(13-i+1)!}\right)^4$ possibilities.

The number of possible observations after the i^{th} trick is:

$$\frac{(52-2i)!}{26!} + (13-i+1)^4 + \left(\frac{13!}{(13-i+1)!}\right)^4$$

Here, the assumption is that any card can be played at any time. But rules make that number significantly decrease, because when a player opens a trick with a certain

suit, other players have to follow with that same suit if they can. Also, in order to win, players play certain cards in certain situations, making the generation of states more deterministic.
At first, the temptation could be to feed the algorithm with as much information as possible, in order to make it better. But such amount of details as input cause issues: the q-value $Q(s_t, a_t)$ for any t, is going to be updated after initialization with very low probability, because s_t rarely appears as it is almost unique.

4.2.2 State Representation Optimization

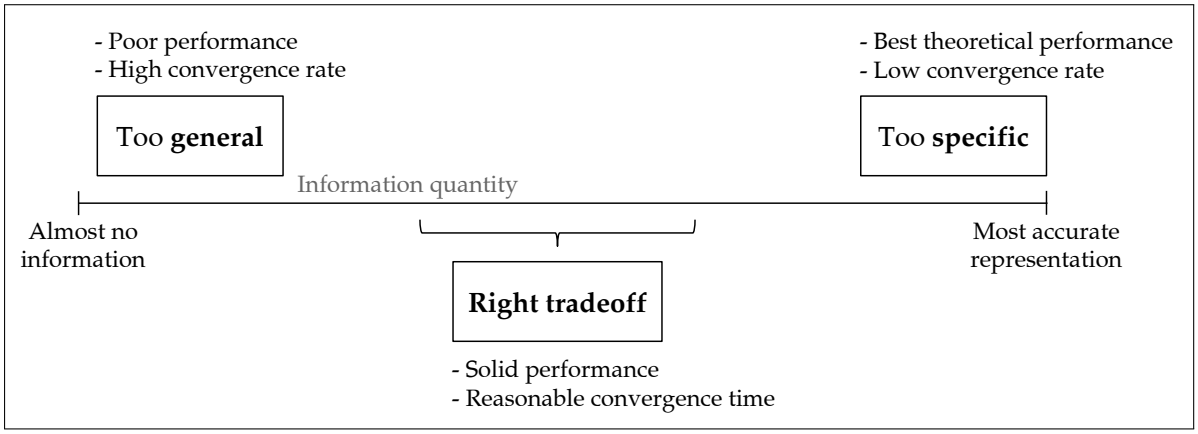


FIGURE 4.2: Comparison between information quantity (horizontal axis) and performances. Finding the right tradeoff in state representation is key.

Information must be filtered and carefully selected.
As listed in Section 4.2.1, an example state representation after the 1^{st} trick could be:
• [2♣, 5♣, 10♣, K♣, 5♢, 6♢, 8♢, Q♢, A♢, 8♡, 6♠, J♠] (`Declarer's hand`)
• [4♣, 7♣, 3♢, 9♢, J♢, K♢, 2♡, 7♡, 9♡, 10♡, 10♠, Q♠] (`Dummy's hand`)
• [N: K♠, E: 4♠, S: 2♠, W: A♠] (`Current trick`)
• [] (`Trick history`)

This representation belongs to the "Too specific" area of Figure 4.2. When fed with this kind of input, all RL algorithms struggle to converge and the performances do not improve at any reasonable time.

A basic strategy of bridge is to count the sure tricks. In other words, the declarer looks at his cards and the dummy's, and finds out how many tricks he is sure to win. For example, if he owns A, K, Q, and J of a suit in notrump play, and has at least four cards of this suit in one of his or the dummy's hand, four tricks are assured.
Every advanced bridge player also keeps track of the history, like the state representation that was introduced. The only difference is that only high cards are useful. For instance, when playing first, second, or third, playing Q can be a sure win if the A and K of the same suit have already been played (notrump case).

Such knowledge about the game suggest a high cards-oriented design of observations representation.

4.3 Modified Q-Learning

The focus is to find trends in order to select the best one, and look for more advanced versions of them in the future. Thus, the range of focus is reduced to a subset of the full game, and the structure of RL algorithms is modified to make them computationally faster.

Learning how to play a notrump contract from the declarer's point of view: In bridge, there are three different position at each trick:

- The declarer, who plays his cards and the dummy's, and tries to reach his contract.
- The player on his left, who tries to make the declarer/dummy pair not reach their contract, and plays right after the declarer.
- The player on the right of the declarer, teammate of the previous player, who plays right after the dummy.

The two last players, even if they are teammates, are asymmetrically positioned. It is due to the fact that everyone can see the dummy's cards. The focus of RL agents in this project is to learn how to play as declarer, in any notrump contract.

Favor games which give the best information: Since using a pre-made database of real games would lead to limits (running out of games, and not being compatible with RL algorithms which need big data sets to converge), the selected learning technique is automatic games generation. However, quality of such games is lower, and the agent may converge slower than with actual professionally played games. That is why those games are filtered and weighted:

- **Filter by result**: When teaching an agent how to play declarer, the learning data set only conveys winning games. The reason is that deleting negative examples prevent spending too much time on non-sense type of plays. Also, because the opponents AI is a set of heuristic rules, they are supposed to play with a decent level, so that games randomly won by the random agent are not as easy to get as loses, but represent interesting data to learn from.
- **Weight by win margin**: Reward is always given at the end of the playing phase (when players all have zero cards). The 100 points limit is not taken in account, because it would imply taking care of the bidding phase too.

  Bridge is a widely spread game, and variants exist. One of them is called Mini-Bridge: it is a shortened form of the actual game, where the bidding phase is simplified. The declarer is automatically determined by the dealing phase. He is the player, among the pair which has the highest combined HP, who

has the highest HP. Before the playing phase, he decides on which contract he wants by looking at his cards and the dummy's cards. Usually, this game is introduced to new players, and there are lots of recommendations in the Bridge literature on how to decide on the contract.

The contracts assigned to the generated games are determined thanks to criteria which are introduced in such tutorials (see Table 4.1).

During the generation of games, when one of those satisfy the result condition, the reward associated to this game is proportional to the winning margin. For example, if the declarer and the dummy have 25 honour points in a notrump scenario, and win 10 tricks to 3, the actions made during this game are fed to the learning agent with a reward equal to $1 \times constant$, because the contract corresponds to 3NT, and they exactly won 10 tricks. If the result is 11 tricks to 2, in the same situation, the reward becomes $2 \times constant$.

Honour points	**21-22**	**23-24**	**25-26**	**27-28**	**29-30**	**31-32**	**33-36**	**37+**
Notrump	7 (1NT)	8 (2NT)	9 (3NT)	10 (4NT)	11 (5NT)	11 (5NT)	12 (6NT)	13 (7NT)
Trump (♣♢♡♠)	7-8	9	10	11	11	12	12	13

TABLE 4.1: Table of correspondence between honour points and recommended contracts in Mini-Bridge[1]

Algorithms for games data set generation: The exploration phase of the RL agent is divided into two main parts. First, games are generated and selected based on their result (previous paragraph), and the stored in a database (Algorithm 1).
The next step is the actual exploration: games are loaded and the agent learn from them (Algorithm 2). The result is a trained agent (Figure 4.3).
This is equivalent to an exploration phase with ε equals to 0.

The Q-table modification: While the generation of games is constant in time and fast, the learning phase can quickly become an issue. For each game (sequence of observations) fed to the q-agent, the learning become slower, because for the i^{th} game learned, the agent needs to remember his Q-table resulting from all previous games. Thus, there is no way to use parallel computing with RL algorithms. As memory grows, it becomes slower to update. What's more, in the case of Q-Learning and SARSA, matrices are required (see Table 4.2), and it is costly in terms of memory.

A hash table representation is better in term of memory cost (Table 4.3). It also has a constant cost for inserting and accessing. For example, in Table 4.2, adding a pair `(state, action)` where `state` is new would also create 51 other cells, for all other cards. With the hash table, only one memory cell is needed and other actions are added later on, only if they appear for the specific state in the training set.

[1]Mini-Bridge tutorial: `https://www.nofearbridge.co.uk/minibridge/beginners_booklet.pdf`

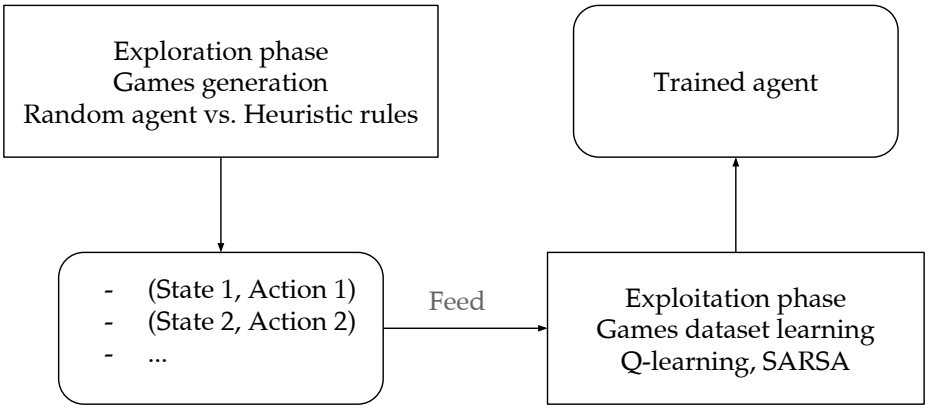


FIGURE 4.3: Framework of a RL agent learning to play bridge.

How to cover everything? So far, the brain of the agent is trained thanks to the generated data set. Also, the goal is to find the right tradeoff between game details and convergence speed. When the agent is in its trained form, playing means searching in its Q-Table for the state, and find the action that maximize the score. If among its current cards, no action is available, the agent has to play randomly. The agent strength depends on the coverage of all possibilities.

In order to maximize the amount of details and keep a large range of known actions, information in each pair `(state, action)` is multiplied.
For example, if (s_t, K♠) leads to winning, positive reward is also given to (s_t, Q♠) and (s_t, A♠), and in some cases to (s_t, J♠) too. Of course, cards that are too different are ignored. Also, the amount of reward given to close cards is decreasing as the distance from the original card grows. The model is as follows:

- For K♠, the agent learns the sequence (s_t, K♠, `reward`, s_{t+1}).
- For Q♠ (respectively A♠), the agent learns the sequence (s_t, Q♠ (respectively A♠), $\gamma_{spread} \times$ `reward`, s_{t+1}).
- For J♠, the agent learns the sequence (s_t, J♠, $\gamma^2_{spread} \times$ `reward`, s_{t+1}).

Algorithm 1: BRIDGE GAMES RANDOM GENERATION

Input: `dataset_size` (size of the games data set).
Output: Generated games data set with consecutive states and actions.

Initialize game data set;
while `dataset_size` > 0**:**
    Initialize a game;
    Initialize the list of states and actions for this game;
    while the game is not done**:**
        Choose a random card from the agent's hand;
        Store the current action (card) and state (before the card is played);
        Update the game based on this card and make the next opponent play;
        Observe the next state;
        Observe the reward;
    if the game is won by the random agent**:**
        Store the list of states and actions of the game in the data set;
        Store the reward and the contract information;
        `dataset_size` $-= 1$;
return *game data set;*

Algorithm 2: AGENT LEARNING HOW TO PLAY BRIDGE

Data: Game data set (pre-generated games with consecutive states and actions)
Output: Trained agent

Initialize RL agent;
for game ***in*** game data set**:**
    Get the reward for this game;
    while the game is not done**:**
        Get the current state, the action, and the next state;
        Feed the agent with the current state, the action, the reward, the next state (and the next action for SARSA);
return *RL agent;*

Q-Table:

	a_1	$\dots$	a_n
s_1	$Q(s_1, a_1)$	$\dots$	$Q(s_1, a_n)$
$\vdots$	$\vdots$		$\vdots$
s_m	$Q(s_m, a_1)$	$\dots$	$Q(s_m, a_n)$

TABLE 4.2: Q-Table default representation.
For each new state s_i, a whole row of size n must be created,
and all $Q(s_i, a_k)$, for $k \in [\![1, n]\!]$, have to be initialized.
In the end, the table is sparse.

$$
\textit{Q-Hash-Table}: \quad
\begin{array}{ll}
\{\ s_1: & \{\ a_1^{s_1}: Q(s_1, a_1^{s_1}), \ldots, a_{n_{s_1}}^{s_1}: Q(s_1, a_{n_{s_1}}^{s_1})\}, \\
 & \vdots \\
s_m: & \{\ a_1^{s_m}: Q(s_m, a_1^{s_m}), \ldots, a_{n_{s_m}}^{s_m}: Q(s_m, a_{n_{s_m}}^{s_m})\} \quad \}
\end{array}
$$

TABLE 4.3: Q-Hash-Table representation. Each new pair `(state, action)` does not involve creating useless spaces.
$a_i^{s_j}$ is the i^{th} seen action for state s_j.
n_{s_j} is the number of actions seen for state s_j.

Results

5.1 Q-Learning and SARSA

Preliminary study: In order to fully understand how to optimize the state representation optimization, a study on a reduced bridge game has been performed. A simple 12-card game was designed, and Q-learning and SARSA both tested. At this step, games generation and exploration are part of the same process. The importance of this preliminary study is high, because signs of positive results confirm the possible development of the project.

Agents are trained and tested against a random strategy, and on some specific deals (in order to lower the size of the steps set). The observations have the form:
[`declarer's hand, dummy's hand, current trick, trick history`], and all the cards are taken in account. Both algorithms converge to the optimal path to victory in a reasonable time. Q-learning is the fastest: it consistently finds the the winning moves after it has seen all the possible moves once. SARSA is ten times slower, probably due to the regrouping of generation and exploration: the reward is only known in the end, so that randomly exploring actions makes the positive values of the Q-table slow to spread. Q-learning updates its q-values based on the optimal best next move, so that reward value spread faster.

Performances and details for Q-Learning and SARSA: For the full game study, using the methods described in the previous chapter, several states representations were tested.

Overall, the human experience of bridge is useful to understand which representation is the best: high cards-focused observations outperforms others when it comes to convergence time. Adding lower cards to the states does not make performances better. It only increases convergence time.
Two kinds of state representation are interesting in term of performances and quantity of information.

- Known hands:

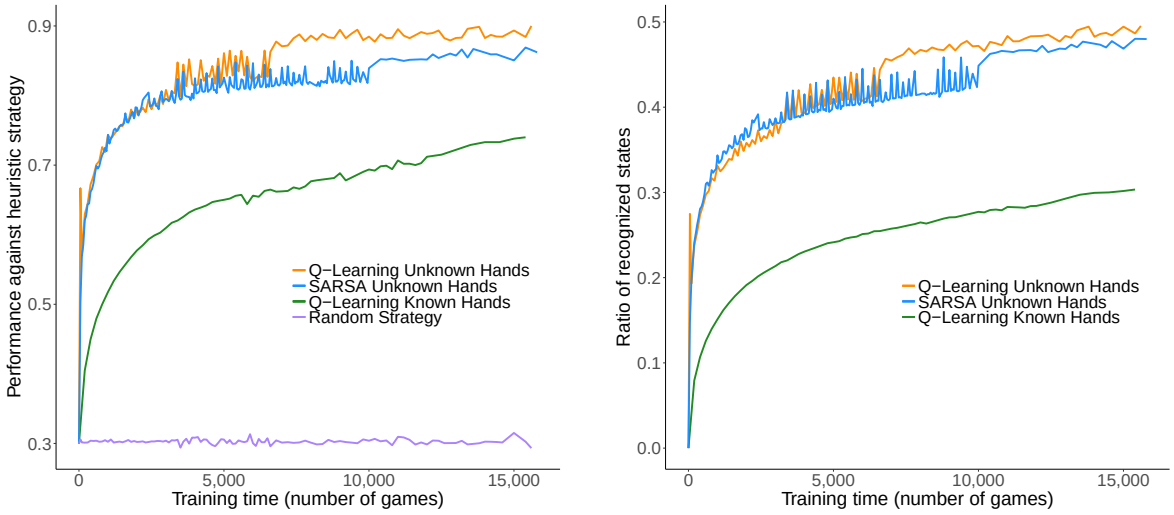
  [`declarer's and dummy's hands, current trick, trick history`], keeping the full information for the current trick, and only taking care of K, and A for the hands, and Q, K, and A for the trick history.

  Example: [[K♣, A♣, K♢, A♡, K♠], [-, -, -, J♣], [K♡, Q♡]] means that the agent's (who plays as declarer) team owns five "important" cards (K and A), only one play played in this trick so far (W played J♣), and in the previous tricks, two "important" cards were played (among all Q, K, and A).

- Unknown hands:

  [`current trick, trick history`], keeping the full information for the current trick, and only taking care of Q, K, and A for the trick history.

  Example: if the game is in the same exact state as in the previous example, [-, -, -, J♣], [K♡, Q♡]] becomes the observation. The agent loses the knowledge of his own cards.



(A) (B)

FIGURE 5.1: Q-learning & SARSA agents' ratio of victory compared to a random play style, and the proportion of recognized states through the learning phase.

Results (Figure 5.1a) are shown for Q-learning and SARSA, for the following parameters: learning rate $\alpha = 0.3$, discount factor $\gamma = 0.9$, and spread coefficient $\gamma_{spread} = 0.8$. The defending pair (declarer - dummy's opponents) is playing according to the previously mentioned rule-based AI. A random declarer agent is struggling to win against this AI (30% win rate): it makes sense, and also shows that the balance between a chosen contract and the declarer's pair HP is well defined.

State representation makes a bigger difference on convergence time and performances than RL algorithms. Q-learning and SARSA seem to have the same convergence rate and maximum performances. Their level is also strongly correlated with the number of states they know. In parallel with the performance testing, the ratio of known states over the total number of states of the test set (1,000 games repeated and averaged) is added (Figure 5.2a). In addition, the total training phase (15,000 games) takes around 30 hours.

There seems to be a slight difference between the curve for each state representation. Even though the known hands configuration creates more states (harder to reach a high ratio of known states), the effect on performance is bigger: "Q-Learning Known Hands" performance keeps a steady growth after 7,500 games seen, while "Q-Learning Unknown Hands" and "SARSA Unknown Hands" struggle to get higher winning percentages.

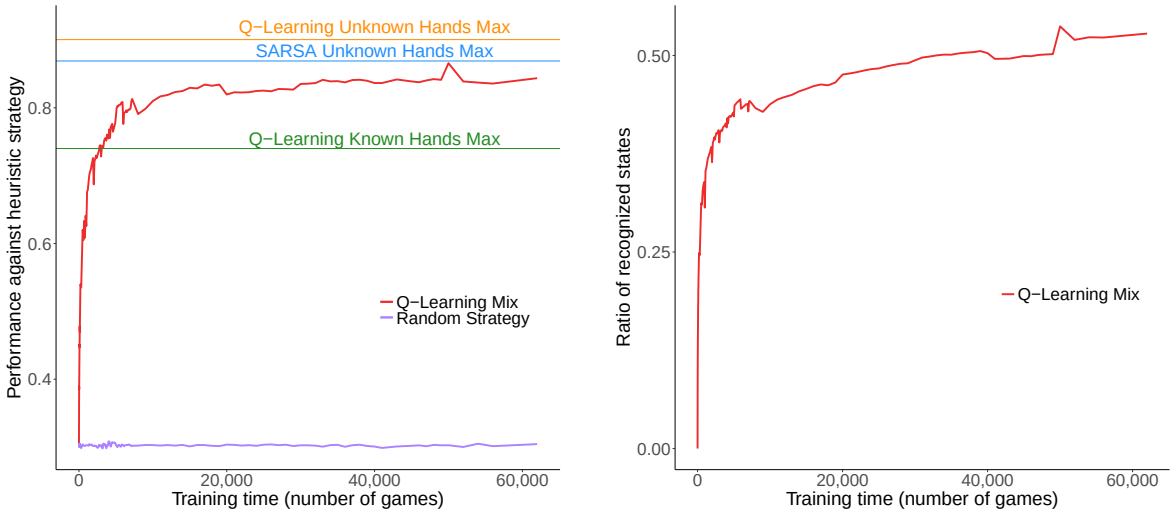
There are two complementary interpretations:

- RL agent ultimately needs more information than in "Q-Learning Known Hands" method if a higher level is the goal. With a longer training phase (around $100,000$ games), "Known Hands" types of methods outperforms the previous ones.
- Training against heuristic strategy brings an inevitable threshold to the playing level of any RL-agent. Even if contracts are efficiently set, some extreme card deals may bring an impossible configuration for the agent to win. Both state representations would converge to the same winning rate (around 90%).

The irregular growth for the SARSA and Q-learning curves (blue and orange) before 10,000 games is tricky to interpret. The reason could be bias: even if performance metrics are calculated several times on different learning sets and then averaged, the fact that the average is made over a small number (around 10) of different learning events could be the cause. The relationship between each spike could happen because of the fact that small learning samples are used to build bigger ones, so that the tendency of maxima and minima remains.
The stop of this irregular growth also introduces a question mark. One probable speculation could be that the subset that makes the spikes is not used anymore in the learning phase. Also, it does not seem that it is correlated to the state space: the "Known Hands" configuration does not show such irregularities. The green curve was generated thanks to the same procedure as the others, so it is hard to conclude for the bias argument.

The spread technique (developed in the last paragraph of the *Methods* chapter) helps to get a higher convergence rate. However, the information added at each learning step is often useless. For example, in both state configurations, the cards added by the spread could already be part of the played card. Example: if the agent plays K♠, and A♠ in in the history, learning to play A♠ does not make sense. Such anomaly does not affect the exploitation phase, since the agent always selects the card in his hand which provides the highest q-value.

Combining information: The main issue about Q-learning convergence is dependence: in order to get an agent which knows 100 states, a 99-state agent is need. To counterbalance the lack of computational power and the unfinished convergence of such methods, different q-tables are combined. New information is added as it is, without modifying q-values. However, a problem occurs when two q-tables have a different q-value for the same pair (s_t, a_t). To cope with that, average values are taken, and each value is also associated to the number of time it is combined, in order to compute averages.
This technique, applied to the same setup as "Q-Learning Known Hands", and with q-tables made from less than 2,000 games, is less efficient. Unlike figure 5.1, the curve on figure 5.2b is steeper than on figure 5.2a: the amount of information that each $Q(s_t, a_t)$ conveys for the methods "Q-Learning Mix" is lower than for the previous methods. In other words, given the same number of states, "Q-Learning Known Hands" makes a better use of them than "Q-Learning Mix".



(A) (B)

FIGURE 5.2: Combinaition of Q-learning agents' performance, and the proportion of regonized states through the learning phase.

Play style and level of the best agents: The level of play of the rule-based agent (opponent) is below the average human level: it is beatable around 80% of the time (tested with me, a beginner-average level player). The performance of the agent for declarer play and notrump contract seems to be acceptable. The play style of best agents consists of focusing on high cards first, and playing low cards when the teammate has played a high one. Such strategy is often introduced in bridge books. It also does not risk wasting high cards if it is not present in the history.

5.2 Critical look - How to improve it?

5.2.1 Critical angles

Can these results be extended to other possibilities of the game of bridge?
Even if the whole analysis is focused on notrump play, RL allows flexibility when it comes to the kind of tasks. For a trump contract, the current agents would probably not perform well, because the rules are too different, and they would not know how to outplay their opponents.
Implementing such an agent involves understanding the stakes of trump play. State representation would have to be modified: in theory, the important cards of notrump situations remains, and the cards of the suit which correspond to the contract are all very important (not only the strong ones), because in some cases, they can beat any other cards.

Playing as defenders is also interesting, since each player of the defending team sees the game differently (asymmetrical views due to the cards of the dummy). One agent for each of them need to be trained. Challenging issues emerge: in which order are they trained? Is training both of them together working?

When teaching the agent to play against a heuristic technique, the learning samples are biased. This technique limits the played cards in specific situations. What if the trained agent meet an unknown state?
Generated games are biased, but so are real games. Bridge players play coherently and do not make absurd moves. Implementing heuristics rules mimics real-life play style. Because it is hard to implement an advanced heuristic strategy, these rules are limited and random play is done when the situation is ambiguous (for instance, if winning at 100% cannot be achieved). All in all, such semi-advanced opponents bring less bias than top players, and cover advanced play styles too.

Why learning with winning games only?
Winning games brings the most information: the goal is the find the right moves, so that feeding the agent successful ways the win has to be part of the process. However, negative examples could also help to learn how to not make errors.
The choice of putting aside this part of the games set is made because of computation time: adding more games increases the learning time. Also, when converged, a RL agent knows, for each state, the best converged value for actions. The most efficient usually has the highest q-value. In theory, the best action at each step should be the same whether negative examples are added or not.

How to increase the performance threshold introduced by the rule-based opponent?
One of the next step of this project is to push the performances of agents further. In order to do so, heuristics can be improved. It implies knowing the right play to do in more diverse situations, and taking care of not making the rule-based opponent more biased than a real player.

Letting agents train themselves against one another sounds like a more flexible way (Figure 5.3), which enables theoretical limitless level. The main concern about such technique is being stuck in local optimal strategies, similarly to those in the optimization field. To counter this, a solution is to introduce more randomness to the learning phase: for example, training a random agent against a trained agent until it outperforms it, is a decent strategy. An ε-greedy RL agent, with ε equal to around 0.5, keeps the previous information in mind and still tries to improve sometimes.

5.2.2 Deep Q-Learning

Applying DQN to the same sample size as the previous algorithms seems to be the next big step of this research. For now, it has shown basic results on small learning data sets: it has a better win rate than a random agent when playing against the same heuristics as the one introduced previously (42% win rate on average when trained with 1,000 games).
Frameworks and parameters have to be built and tuned efficiently. Since it needs the use of specialize libraries (Tensorflow, Keras), algorithm customization, as well as the usage of theoretical and structural knowledge of the techniques is not a easy as before.

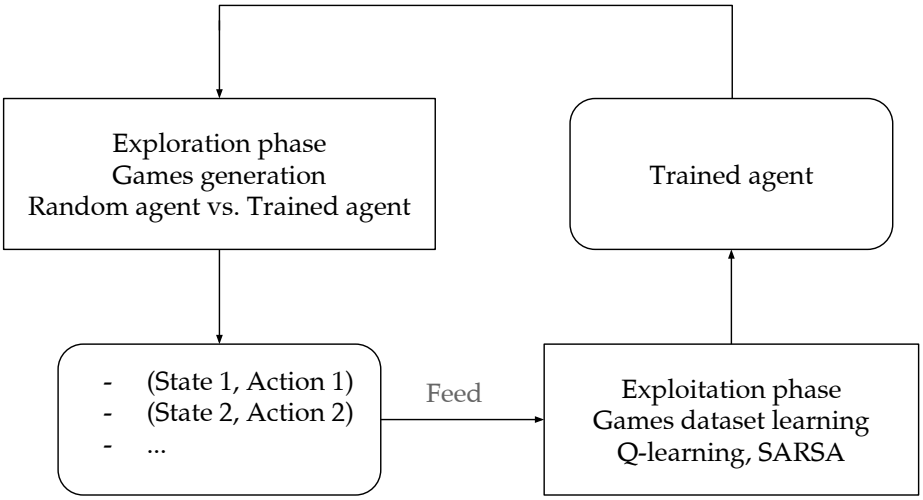


FIGURE 5.3: Updated framework for the bridge learning phase: multi-agent training.

DQN is promising, because its main feature is to translate the states from discrete to continuous. Even if a state is unknown, an estimation of the best move can be performed thanks to similar known states.

Personal Analysis

6.1 Discovering a Japanese research environment

Performing research in a Japanese laboratory is challenging: the habits and the language are different from what I was used to. With the time, adaptation helped me to be more efficient performing my own work. I am comfortable with English and have a good working knowledge, but there is always a delay when it comes to fully understand technical vocabulary.

I had the chance to earn a lot of different scientific pieces of knowledge, as the laboratory holds some seminars and journal clubs every week. Seminars allow to have an idea of what others do in the lab. They are wide in term of topics, and teach a lot about how a research topic is tackled. It always gives new ideas.

6.2 Main Challenges

Since the project was performed on my own, with Prof. ISHII and Mr. NAKAE as supervisors, I had to be proactive and take initiatives. In my opinion, communicating with others, asking for details instead of loosing hours looking alone for inaccessible information, and crossing opinions and ideas in order to move on effectively are the most important things to remember.
Organization is also key. Working most of the time alone implies having a clear idea of what must be done and what the future plans are. This is the first time that I faced such a configuration, and I learned a lot from it: I used personal organization tools (Trello, for instance) in order to stay up-to-date.
As the working environment is a laboratory, objectives are more blurred than in a company environment. The goal is to go as deep as possible into the theme. In a research laboratory like here, the more mistakes and useless results are made, the more interesting results and conclusions will occur.
Overall, this helped me a lot to build my thoughts concerning my future career.

The setup of this project is atypical, because I attended classes through the whole year, so that the research was performed half of the time.
Having two main things to deal with is not an easy task, especially when one tends to take over the other. For example, when class projects are happening, laboratory research's time has to be reduced. Then, when coming back to the research, it is challenging to dive back in it and to get to cruising speed. In general, it helped me to find ways to stay motivated.

Conclusion

Most games AI researches focus on reinforcement learning based techniques. They enable task flexible, thanks to their model-free structure. However, most of the currently mastered games are board games: imperfect information games are still an issue for computers.

Here, reinforcement learning methods are applied to contract bridge in order to compare them, and look for the key points of the game. Focusing on some details rather than others makes performances increase considerably, whereas giving too much details to the agent slows down the convergence time. Strong cards are the most important features for observations, and not showing which cards the agent has does not affect performances in this setup. The learning framework is also customized. Games are pre-generated and filtered, so that rewards are known at the start of the games, saving q-value spreading time (especially for SARSA). Covering the state space is also facilitated thanks to the hypothesis that similar cards offer similar outcomes. Finally, the fact the implementation is flexible allows the modification of q-tables' structure, saving memory and time.

Competing against advanced human players still seems far, but current results show promising improvements and establish a basic for the future of this research. Adding more advanced techniques like deep learning could help to cover even more states. Future plans also include involving several agents in the learning phase, and start learning with a better rule-based agent.

Summary (French)

Ce projet de fin d'études vise à explorer des méthodes d'intelligence artificielle appliquées au bridge. Ce jeu de carte est un défi pour les méthodes actuelles puisqu'il met en scène deux paires de joueurs qui s'affrontent, et au sein desquelles les joueurs doivent coopérer tout au long de deux phases de jeu afin de l'emporter. Cet aspect, couplé à l'incertitude qu'introduisent les cartes cachées, présente plus de difficultés que les jeux de société déjà maîtrisés par les algoritmes les plus récents (jeu de Go avec AlphaGo Zero de Google en 2017).

L'apprentissage par renforcement, branche de l'apprentissage automatique (ou machine learning), consiste à laisser évoluer un agent intelligent dans un environnement inconnu. Il y apprend les mécaniques, et comment adopter un comportement optimal. Un des avantages de cette méthode est l'absence de modèle. La phase où l'agent explore et exploite librement son environnement fait la distinction entre apprentissage par renforcement et autres techniques de machine learning. En effet, ces dernières se concentrent pûrement sur de l'exploitation de données.

Q-learning et SARSA sont les principales méthodes utilisées. Pour les deux, l'agent associe, pour toute configuration, un score à une action. En répétant ce processus, il trouve vers les meilleures actions pour chaque situation (i.e., celles qui maximisent la somme des scores).
Q-learning requiert de recevoir une observation, une action, l'observation résultante, et une récompense calculée en fonction de la performance liée à l'action effectuée. Le score pour cette paire action/observation est ensuite mis à jour en fonction de la récompense, et du score maximal futur associé à l'action résultante.
SARSA a pour différence d'ajouter l'action suivante à la liste des variables que reçoit l'agent. La mise à jour du score pour l'action et l'état courants requiert alors la récompense, mais pas le score maximal futur: à la place, le score pour l'action suivante est prise en compte.

Ces méthodes sont légèrement ajustées et les observations sont modélisées avec soin afin d'avoir un agent qui converge rapidement et qui joue avec un niveau correct. Les cartes observées sont notament sélectionées de sorte à ne garder que celles qui font directement gagner des manches.

Les résultats montrent que les agents d'apprentissage par renforcement obtiennent des performances convenables après un temps d'apprentissage raisonnable. La nature de ces résultats amènent plusieurs questions d'interprétation et de nouvelles problématiques en émergent.

Ce project permet d'ouvrir des perspectives pour l'apprentissage par renforcement

du bridge. Améliorées et couplées à l'apprentissage profond (deep learning), les méthodes actuelles peuvent prétendre à un niveau de jeu similaire à celui d'un joueur humain expérimenté.

Bibliography

Berlekamp, Elwyn R. (July 1963). "Program for Double-Dummy Bridge Problems&Mdash;A New Strategy for Mechanical Game Playing". In: *J. ACM* 10.3, pp. 357–364. ISSN: 0004-5411. DOI: 10.1145/321172.321182. URL: http://doi.acm.org/10.1145/321172.321182.

Berliner, Hans (Jan. 1980). "Backgammon Program Beats World Champ". In: *SIGART Bull.* 69, pp. 6–9. ISSN: 0163-5719. DOI: 10.1145/1056433.1056434. URL: http://doi.acm.org/10.1145/1056433.1056434.

Fujita, Hajime and Shin Ishii (2007). "Model-Based Reinforcement Learning for Partially Observable Games with Sampling-Based State Estimation". In: *Neural Computation* 19, pp. 3051–3087.

Goodfellow, Ian, Jonathon Shlens, and Christian Szegedy (2015). "Explaining and Harnessing Adversarial Examples". In: URL: http://arxiv.org/abs/1412.6572.

Heinrich, Johannes and David Silver (2016). "Deep Reinforcement Learning from Self-Play in Imperfect-Information Games". In: *CoRR* abs/1603.01121. arXiv: 1603.01121. URL: http://arxiv.org/abs/1603.01121.

Ishii, Shin et al. (2005). "A Reinforcement Learning Scheme for a Partially-Observable Multi-Agent Game". In: *Machine Learning* 59.1, pp. 31–54. ISSN: 1573-0565. DOI: 10.1007/s10994-005-0461-8. URL: https://doi.org/10.1007/s10994-005-0461-8.

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (May 1998). "Planning and Acting in Partially Observable Stochastic Domains". In: *Artif. Intell.* 101.1-2, pp. 99–134. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(98)00023-X. URL: http://dx.doi.org/10.1016/S0004-3702(98)00023-X.

Schaeffer, Jonathan et al. (2007). "Checkers Is Solved". In: *Science*, pp. 1144079+. DOI: 10.1126/science.1144079. URL: http://dx.doi.org/10.1126/science.1144079.

Silver, David et al. (Oct. 2017). "Mastering the game of Go without human knowledge". In: *Nature* 550, pp. 354–. URL: http://dx.doi.org/10.1038/nature24270.

Smith, Stephen J. J., Dana S. Nau, and Thomas A. Throop (1998). "Computer Bridge - A Big Win for AI Planning". In: *AI Magazine* 19.2, pp. 93–106. URL: http://www.aaai.org/ojs/index.php/aimagazine/article/view/1371.

Sutton, Richard S. and Andrew G. Barto (1998). *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press. ISBN: 0262193981.

Ventos, V. et al. (2017). "Boosting a Bridge Artificial Intelligence". In: *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1280–1287. DOI: 10.1109/ICTAI.2017.00193.

Ventos, Veronique and Olivier Teytaud (June 2017). "Le bridge, nouveau défi de l'intelligence artificielle ?" In: *Revue des Sciences et Technologies de l'Information - Série RIA : Revue d'Intelligence Artificielle* 31.3, pp. 249 –279. DOI: 10.3166/ria.31.249-279. URL: https://hal.inria.fr/hal-01665780.

Watkins, Christopher J.C.H. and Peter Dayan (1992). "Technical Note: Q-Learning". In: *Machine Learning* 8.3, pp. 279–292. ISSN: 1573-0565. DOI: 10.1023/A:1022676722315. URL: https://doi.org/10.1023/A:1022676722315.

Watkins, Christopher John Cornish Hellaby (1989). "Learning from Delayed Rewards". PhD thesis. Cambridge, UK: King's College. URL: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

Yeh, Chih-Kuan and Hsuan-Tien Lin (2016). "Automatic Bridge Bidding Using Deep Reinforcement Learning". In: *CoRR* abs/1607.03290. arXiv: 1607.03290. URL: http://arxiv.org/abs/1607.03290.