Counterfactual Regret Minimisation for playing the multiplayer bluffing dice game Dudo

Quentin Gendre[1,a)] Tomoyuki Kaneko[2,b)]

Abstract: Counterfactual Regret Minimisation (CFR) is a powerful learning tool for making AIs for incomplete information games, but the vanilla algorithms becomes impractical when confronted with large size problems. In this paper, we propose to explore the capabilities of CFR-based algorithms on the test case of 4-player Dudo, a bluffing dice game often used in the 2-player case, but hadn't yet in multiplayer CFR research. To curb the overwhelming amount of states and depth of the game, we combined two methods: The abstraction through imperfect recall, thus reducing the size of the game space; and CFR training variations, such Pruning, Monte Carlo and/or Fixed-Strategy Iteration. After implementing various combinations, some had very satisfying results when tested against hand-made AIs, and Pruned Fixed-Strategy Iteration – a training method introduced in this paper that combined existing ones – returned a very encouraging insight on how training time can be reduced.

Keywords: Counterfactual Regret Minimisation, CFR, multiplayer, dudo, perudo, bluff

1. Introduction

Counterfactual Regret Minimisation (CFR) has proven itself to be a good robust base-line for approximating Nash Equilibrium for 2-player zero-sum incomplete information games, thus becoming a common starting point for 2-player poker-like games. In these 2-player zero-sum cases, the Nash Equilibrium also has very interesting properties, such as guaranteeing the player using it a non-negative average utility (namely to "not lose"), regardless of the strategy of the other player [7]. Various algorithms using CFR as a base-line have emerged to find this equilibrium efficiently, in poker or other incomplete information games[4].

However, making AIs outside of these constraints is still an open problem. Other than the challenge of no longer having an absolute solution, there also are difficulties such as the potential explosion of the size of the state and action space. There has been some research conducted on poker with more than two players, that show that CFR remains efficient in environments that relax its original conditions. Typical examples are 3-Player Kuhn Poker [9], 3-Player Leduc Hold'em [9], and multiplayer Limit Texas Hold'em [5]. These research show that CFR applied to multiplayer environment looks promising, although some extra care is required as Nash Equilibrium doesn't guarantee a positive average utility anymore. A very recent study has even managed to make a very powerful AI against humans in six-player no-limit Texas hold'em poker [2].

Dudo – also known under different names such as Cacho, Pico, Perudo or Liar's dice – is an old bluffing dice game still popular in South America. There exists a lot of variations to this game, but in all of them, players must make in turn higher and higher claims of type "there is a certain amount of dice with a certain value", until one or more player doubts them, in which case the trustfulness of the claim is verified and the liar(s) eliminated. It shares some similarities to poker, randomness and bluffing , thus making it a great candidate for testing CFR. 2-Player Dudo has already been a test environment for CFR [3], [8]. However, to our best knowledge, no research has tried using multiplayer CFR for the game Dudo, although some research has already been done using a different type of algorithm "Soar" [6]. Apart from the even larger state and action space, 4-Player game Dudo offers some additional challenges due to the break of symmetry. Since there is a single player claiming, followed by three players doubting, the claim and doubt actions can no longer be combined, the actions are no longer homogeneous and the players won't always take actions in the same order.

In this paper, we propose to explore the potential of CFR-based algorithms on the 4-player game Dudo. Different candidates for algorithms and abstraction have been implemented, tested, and compared using base-line AIs. Most of these agents are adaptations of algorithms already tested for 2-player Dudo, and one of them introduced in this paper – Pruned Fixed-Iteration Strategy Counterfactual Regret Minimisation – was inspired by combining two others.

2. Environment

2.1 Rules of Dudo

In the most simple rule of Dudo, each player will roll 5 standard 6-sided dice, look at the result and keep it hidden from the other players. Then, players will each in turn make a "claim". These claims are a die value (1 to 6) and an amount of die representing a statement "there are *amount* dice of *value*". After each statement, opponents can challenge the last caller, or keep going

[1] Graduate School of Interdisciplinary Information Studies, the University of Tokyo
[2] Interfaculty Initiative in Information Studies, the University of Tokyo
[a)] gendre@g.ecc.u-tokyo.ac.jp
[b)] kaneko@graco.c.u-tokyo.ac.jp

with the claims. If challenged, everyone shows their dice and the dice with the same value as the claim are counted. The caller loses if there are collectively less dice with the correct value than the amount claimed. Otherwise, if there are as many or more dice than the claim, it is the player(s) that challenged the claim that lose. The loser or losers are eliminated from the game, and if there are two or more player left, the game continues. [*1]

In order to force players to take risk, and prevent the game from going in circles, there is an additional rule : Claims have to increase either the "value" or the "amount" of the previous claim, and the amount can not be decreased.. For example, after calling 2×4 (ie. 2 dice of value 4), you can call 3×2, 3×3, and 3×4 (because the "amount" has been increased); 2×5 and 2×6 (because the "value" has been increased), but not 2×3 or 2×4 (neither the "amount" nor "value" has been increased) nor 1×5 and 1×6 (the "value" was increased but the "amount" was decreased).

2.2 Restrictions in this paper

Our paper added some restrictions on the game, in order to better fit to the computer's specificities, and simplify training and testing.

2.2.1 Four players, Single Round

Since we are looking at the specificity of a multiplayer game, we want to focus our AI on the 4-player case. Therefore the environment of the paper is a 4-player game of Dudo that stops after the first round. Since a final winner hasn't necessarily been determined, the players remaining will receive a reward corresponding to one divided to the amount of remaining players – namely $\frac{1}{3}$ each if 3 players are remaining, $\frac{1}{2}$ for 2 players, and 1 for 1 player.

2.2.2 Turn based

In the original game, challenging the claimer is done in real-time. The first player to say "dudo" will usually be the only doubter, unless someone else speaks at the same time. This is a moment where there is supposed to be tension, as every player (except the claimer) hope someone else will challenge the claim before being next.

However, this dimension was completely removed in the computer implementation. In real-time the optimal solution would of course be waiting until the last moment, and then answering. This isn't interesting from an Artificial Intelligence point of view. Therefore in our environment, each AI answers if they would like to challenge the claim or not, without knowing the actions of the other players. This can be done in parallel. The end of the turn is resolved normally.

Furthermore, CFR requires a turned base game, so a claim & challenge turn was cut down in 4 smaller steps :

- Player *A* makes a claim
- Player *B* decides if she wants to challenge the claim
- Player *C* decides if she wants to challenge the claim
- Player *D* decides if she wants to challenge the claim, and players *B*, *C* and *D*'s intentions are revieled. If at least player challenged the claim, the claim is resolved and the game ends. Otherwise restart after rotating the players.

[*1] In some other rules, the loser can lose only one die, or loses a variable amount of dice depending on how far his claim was from reality.

2.2.3 Limiting Claim Space

In a 4 players 5 dice game, there are 20 dice in total, and 6 different values (1 to 6). Therefore there are a maximum of 120 different possible claims, all of which are supposed to be available at the start of the game.

However, in the overwhelming majorities of games, claims are always close to the previously announced claim. In order to simplify the environment without altering the rules too much, the allowed possible claims are being limited to the next 18 (order of strength). In other words, the "amount" of the claim can be increase by 3 at most. For example, if the previous claim was 2×4, the next claim has to be selected between 2×5 and 5×4 (included).

Table 1 Probabilities of rolling identical dice when rolling 20 dice

Identical dice	Probabilities
5	92.85%
6	55.41%
7	21.95%
8	6.74%
9	1.70%
10	0.36%
11	0.063%
12	0.0092%
13	0.0011%
14	0.0001%

Furthermore, the probability of there being more than 11 dice of the same value is very low: 0.063% (Table 1). Therefore we decided to force our AIs to challenge any call with an amount higher than 11, regardless of the situation. This divides the maximum depth of the game tree by 2, increasing training speed, while diminishing our win-rate by an insignificant amount.

3. Training Agents

This paper will be first presenting the theory of Counterfactual Regret Minimisation, followed by descriptions of the different optimisations and variations tested in this research.

3.1 Counterfactual Regret Minimisation

In Counterfactual Regret Minimisation (CFR), the algorithm trains agents through self play. At each training iteration, it will improve its current strategy by improving the probability distribution of one situation, comparing one player in opposition to the others. It is important to note that this is possible in CFR as the training algorithm knows perfectly the behaviour of every agents. The following notations are strongly inspired by [8].

We start by defining the counterfactual value $v_i(\sigma, h)$ for player i, at a state h, following strategy σ as:

$$v_i(\sigma, h) = \sum_{z\in Z} \pi^{\sigma}_{-i}(h) \times \pi^{\sigma}(h, z) \times u_i(z) \qquad (1)$$

Where Z denotes the terminal states, $u_i(z)$ the utility for player i at terminal state z, $\pi^{\sigma}(h, z)$ the probability of reaching state z starting from h and following strategy σ, and $\pi^{\sigma}_{-i}(h)$ the probability of reaching h without taking player i into account. In other words, $\pi^{\sigma}_{-i}(h)$ represents the probability of reaching state h having every player follow σ, except for player i that will play its actions with probability 1.

The counterfactual regret for not taking action a at state h is defined as

$$r(h,a) = v_i(\sigma_{I\to a},h) - v_i(\sigma,h).$$

This regret represents how much the agent regrets not having picked a, in other words "what would have been my increase in utility had I picked a".

Therefore, if we call I the information set (infoset) of player i at state h, such as player i can not distinguish h from other states of the set I, we define the counterfactual regret of an information set as

$$r(I,a) = \sum_{h\in I} r(h,a).$$

We then update σ so that the probabilities distributions of actions in a information set I is proportional to the cumulative regrets for each action, ignoring actions with negative regrets. If we call $r_i(I,a)$ the cumulative regrets of actions a in an information set I, and $r_i(I,a)^+ = max(r_i(I,a),0)$, we can write the probability of picking action a when in I as

$$\sigma_i(I,a) = \frac{r_i(I,a)^+}{\sum_{a\in A(I)} r_i(I,a)^+}$$

If all actions have negative regrets, namely $\sum_a r_i(I,a)^+ = 0$, it defaults to the homogeneous distribution $\sigma_i(I,a) = \frac{1}{|A(I)|}$.

A more detailed explanation of the algorithm and proof can be found in [7].

3.2 Pass-Through types

One way to implement this theory is using a Pass-Through type algorithm. At each iteration, the algorithm will simulate an initial game state. Then the game will be played from start to finish, but each time a state is reached, the regret of each actions is computed, and the strategy for this state is updated.

However, computing the exact regret would require to have the algorithm iterate over all possible end states, in order to determine the precise probability and utility for reach each of them. This in practice takes too long, so we tested different types of optimisations described bellow.

3.2.1 Pruned Pass-Through CFR (PPT)

One way to shorten the time is pruning branches during the exploration on the tree. This consists in artificially considering some states to be end states, and replace its real utility by an approximation. Therefore the real children end states (and all states in between) won't be explored, saving computational power. However, these *pruned* states must be picked carefully, otherwise the approximated regret will be too far from the real one.

In the Pruned Pass-Through (PPT) used in the paper, branches contributing for less than 10^{-4} of the utility will be replaced by the average utility $\frac{1}{4}$ (because there are 4 players).

3.2.2 Monte Carlo Pass-Through CFR (MCPT)

Similarly to above, this algorithms will estimate the utilities of the different states in encouters during a training iteration. Here, the utility is approximated using Monte Carlo. Namely, a game is simulated until the end multiple times, and the average utility of those games is used as an approximation. Since the probability distributions of every player are known, the average utility is well approximated.

In the Monte Carlo Pass-Through (MCPT) used in the paper, branches are 1000 times.

3.3 Full exploration types

Another way to implement CFR is to generate a initial game state, explore all the possible states of the game tree, and update the strategy for each of these situations. The main advantage of this technique is that resources can be saved by using computed values multiple times, such as when the utilities of the leafs are calculated and propagated back to their parents. Since the regrets for every states have to be computed, and not only one path, one of these iterations takes much longer than one of a Pass-Through type. However, in a proportional way, there is also more learning done.

The variations of this type tested in this paper are described bellow.

3.3.1 Fixed-Strategy Iteration CFR (FSI)

During the exploration of the tree, the agent might encounter states that are very similar. If the states are similar enough, the players would play similarly in both of theses cases, and the next states would also be similar. The natural intuition would be to "merge" these states together, so one would only need to explore them one time in total.

With the vanilla algorithm, this wouldn't be possible, as the strategy of the player keeps on changing as the tree is explored. To counter this problem, the paper [8] propose Fixed-Strategy Iteration: during the whole iteration, the saved cumulative regrets – and so the strategy – remains untouched, and will only be updated completely at the end.

Two states can be merge if the same actions will lead to the same utilities, and if every player would play the same. The rules of Dudo are relatively simple, so the dice of the players, the current claim, and the next actions is enough to completely determine the outcome – no history of the past actions are required. If agents had perfect memory, no state could be merge, as they could be behaving differently. However training a CFR agent with perfect memory is impractical, so we use infoset Abstraction (Cf. Part 4). Now, if these two states have the same abstraction, and the game is in the same configuration, they can be merged. This can be summarised as: If neither players nor the rules can distinguish the states, then they are the same.

A specificity of a game of Dudo is that, since the claims can only go higher, the states of a game can be represented as directed acyclic graph. Namely, the states can be ordered, and taking an action at a certain state can only lead to a state further in the order. For Dudo, the states can be ordered according to the Claim, and the next player's turn (claim, first doubt, second doubt, and third doubt). The algorithm can therefore be efficiently divided into two iterative passes:

- One top to bottom, where the probabilities of being reached will be propagated
- One bottom to top, where the utilities and regrets will be back-propagated

During the top to bottom iterations, the states will propagate on the lists $\pi^{\sigma}_{-i}(h)$ probabilities for each player. (See Part 1 for notations.) Every node only needs the $\pi^{\sigma}_{-i}(h)$ corresponding to their player i (Equtation 1), but they need to transmit the whole list to their children that might need a different one (cf. Algorithm 1).

```
With :
h state of parent;
p current player playing;
$\mathcal{P}\left(\sigma_p : h \rightarrow h'\right)$ probability of player p at state h and following
  strategy σ of choosing an action leading to h';
for each child in state $h_c$ do
    for each player i do
        if $i \neq p$ then
            $\pi^{\sigma}_{-i}(h_c) \leftarrow \pi^{\sigma}_{-i}(h_c) + \mathcal{P}\left(\sigma_p : h \rightarrow h_c\right) \times \pi^{\sigma}_{-i}(h)$;
        else
            $\pi^{\sigma}_{-p}(h_c) \leftarrow \pi^{\sigma}_{-p}(h_c) + \pi^{\sigma}_{-p}(h)$;
        end
    end
end
```

Algorithm 1: FSI probability propagation

Similarly, during to bottom to top iterations, the states will back-propagate their average utilities for every player, which is a list of sum 1. Once a node has all the average utilities of its children, it will compute the regret of the state and add it to cumulative regret.

3.3.2 Pruned Fixed-Strategy Iteration CFR (PFSI)

The FSI algorithm described above will compute and update every state in the game tree. Every branch is explored, regardless of their contribution to the regrets of the parents node, and by how much the update would impact the strategy. Since a lot of these states have actually a very low probability of being reached, we had the idea of combining the Fixed-Strategy Iteration base-line with the Pruning optimisation.

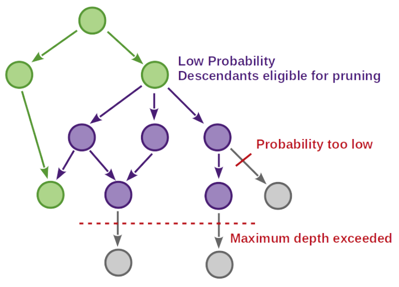

Fig. 1 Representation of Prunning method introduced for FSI
Green : Unflagged
Purple : Flagged
Grey : Unexplored

Our Pruned Fixed-Strategy Iteration (PFSI) algorithm follows a very similar process from FSI, except for the following points :

- During the top to down iterative pass, if a step doesn't have a high enough probability of being reached, the children states will be flagged as "can be approximated". In our case, states with a probability $\pi^{\sigma}_{-i}(h) < 10^{-20}$ have a probability of 0.98 of flagging their children. Once states are flagged, their children can be subjected to pruning.
- During the top to down iterative pass, a flagged state will be more selective on its children, and will continue to propagate flags to their children. [*2] A flagged state will only ask for the average utilities of actions with a probability higher than 10^{-4}, and approximate the rest. Furthermore, after a certain depth of consecutive flagged children, the step will stop exploring deeper and approximate its utility. Approximates utilities are $\frac{1}{4}$ for each player (similarly to Pruned-Pass Through).
- During the bottom to top iterative pass, the states that are flagged will only be computing their utility and back-propagating it to their parents – the regret can not be calculated, and it will not used to update the strategy.

We tests this algorithm with two different sets of parameters :

- PFSI-1 where the flagged exploration depth is 1
- PFSI where the depth is 4

4. Abstractions

Since remembering the previous calls made during a game can offer information, would want to distinguish situations with different histories. However, there is a colossal amount of different possible histories, a the action and state space of Dudo can quickly explode. For a regular game of only 2 players playing with 5 dice each, there are already 10×6 different possible claims, and approximately 3×10^{20} different information sets [8]. For 4-player Dudo, there are 120 different claims, and so 2^{120} different histories. If we combine this with the 252 possible dice a player can have, we would have a infoset size of 3×10^{38}. Tackling a size this big is impossible for current machines.

To curb this problem, we can use abstraction. Each infoset is simplified into fewer characteristics, becoming indistinguishable from similar ones, thus reducing the size of the data to train. However, the more abstracted an infoset is, the more information is lost. The simplest way to do this would be to truncate the previous action history, keeping only the more recent ones: this is called Imperfect Recall.

We present bellow different abstractions used in our tests.

4.1 No Recall (NR)

This is the most radical abstraction : No memory is kept. The only information available to make decisions are the current claim, and the visible dice.

4.2 Type Recall (T)

The history is truncated, and among the previous claims, only the "type" or "value" of the dice is kept, not the amount. In this paper, a memory of 3 is used.

For example, if the current claim is 4×3, and the previous claims were 4×1, 3×4, 3×2, 2×1 and 1×6, the abstraction will keep the current claim (4×3) and the last 3 claim values ($? \times 1$, $? \times 4$, $? \times 2$).

Not saving the amounts helps having a faster FSI algorithms, as many branches can be merged pretty early (Cf. 3.3.2)

[*2] If a flagged state is combined with an unflagged state, the resulting state will be unflagged. The state shouldn't be approximated anymore because at least one parent state needs it.

4.3 Wait & Type Recall (WT)

This is a slight improvement of the previous abstractions. The AI will still remember the last 3 types, but will also know the player's relative position to the current claimer. Namely, it knows how much claim turns it must wait before it is its turn to make a claim.

5. Experimental Results

5.1 Method

5.1.1 Implementation

The algorithms described in Part 3, as well as the Abstractions described in Part 4 were implemented in Python 3.7.4. The numpy library was extensively used to optimise the operations.

We tested the trained AIs on a clean environment: a Python Client/Server-based implementation of the game Dudo.

5.1.2 Baseline AIs

The performance of our agents were tested on by playing against two base-line AIs for Dudo. Due to the lack of canonical AIs for testing performances in Dudo, we implemented the following two using our own knowledge of the game:

- A Random AI. It plays completely randomly, without taking its own dice into consideration, but with weighted probabilities. Its chances of picking each claim is $\frac{1}{9}$ if it's among the first 6, $\frac{1}{24}$ if in the next 6, or $\frac{1}{72}$ if among the last 6. It also has a $\frac{1}{16}$ chance of challenging each claim.
- A Heuristic AI. At the start of the game, once it knows its dice, the AI computes the probability $P(C)$ of each claim C of being true knowing already 5 dice among them. For the rest of the game, when picking a claim, the AI weights its choices by theses $P(C_i)$. When doubting, the Heuristic AI will only challenge the claim if it's the next in line for claiming and with a probability of $1 - P(C)$.

5.1.3 Win-Rates

The performance of agents are always evaluated by pairs. Since this is a 4-player game, multiple positions have to be occupied by the same types of AIs.

When writing A vs B = (??%, ??%, ??%), the displayed percentage correspond to the average win-rates of when playing 3A vs 1B, 2A vs 2B, and 1A vs 1B.

Due to the one round nature of our environment, most games will end up with a $\frac{1}{3}$ score for 3 agents. This means that even very bad players will often indirectly gain scores when other players win, and so the win-rates are always very close to 25%. If the reader wants to be convinced, one can consider an example were one player A is always making the wining move and eliminating another player, and players B, C and D are passive, the final win rates would be approximately 33% for A, and 22% for the others : the huge gap of skills between these agents would be compacted in an 11% difference. Therefore, in order to evaluate the actual performance between the different agents, there needs to have a very good accuracy.

The win-rates displayed in graphs have been determined by using the average score of 100 000 games. For an confident interval of 95%, this guaranties a theoretical uncertainty lower than 0, 32%. Empirically, the uncertainty is actually 0.10%.[*3] These graphs uses the 1vs3 win-rates because they are the ones with the largest amplitudes.

5.1.4 Hardware

The code was executed on a mutlicore server. Since all our training code is sequential, each agent was trained on equivalent single cores, and so their training time can be compared.

5.2 Results

5.2.1 Speed

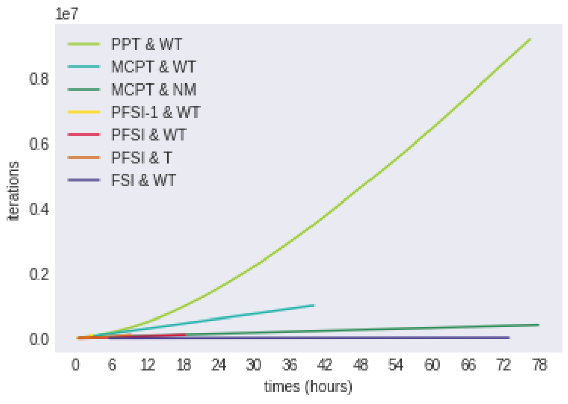

Fig. 2 Iteration speed of the different algorithms

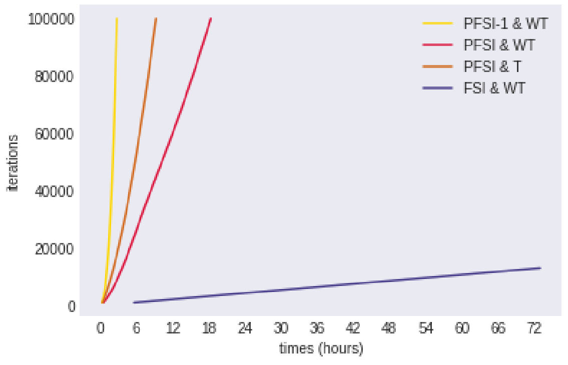

Fig. 3 Iteration speed of the FSI algorithms

Table 2 Average number of iteration per hour

Agent	Speed
PPT & WT	419709
MCPT & WT	589974
MCPT & NM	47431
PFSI-1 &WT	158889
PFSI-4 & WT	63774
PFSI-4 & T	72489
FSI & WT	357

As it can be seen on figure 2, the different AIs have very different training speeds. The speed of PPT for example is more

[*3] $z^* \frac{\sigma}{\sqrt{N}}$ where z^* is 1.96 (because of 95% confidence interval) σ is the standard deviation of the sample, and N is the number of sample. The standard deviation is not really know, but in the worst case scenario, the score would alternate between 0 and 1 with probability p, in which case the standard deviation would be $\sqrt{p(1-p)}$, itself maximised by 0.5.
Empirical test conducted during training show that the standard deviation is actually closer to 0.15, thus having a maximum uncertainty of 0.10%

than 1000 times faster than FSI. Comparing the evolution of their criteria using only the number of iterations wouldn't give much insight on their actual performance. Therefore, graphs comparing AIs of different type will always use time (in hours) as their x-axis.

5.2.2 Completeness

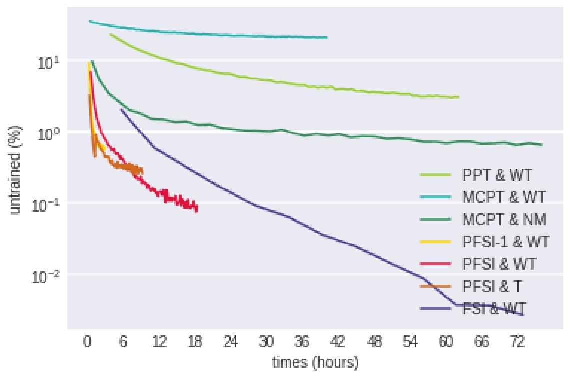

Fig. 4 Evolution of the proportion of untrained states

The above graph represents the speed at which the different CFR agents fill in their data. The "Untrained" is computed during the evaluation part, and represents the proportion of states that where not yet encountered during training.

5.2.3 Win-Rates

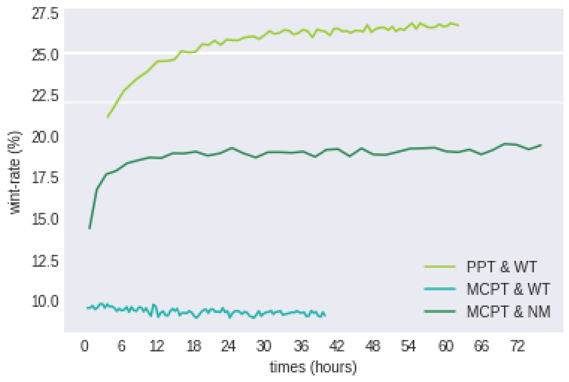

Fig. 5 Evolution of Pass-Through AIs' performance against 3 Heuristic AIs

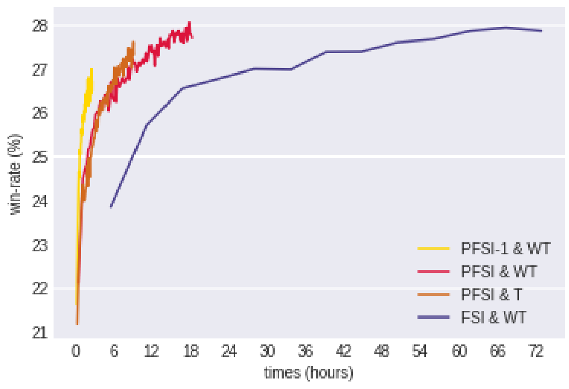

Fig. 6 Evolution of FSI AIs' performance by time against 3 Heuristic AIs

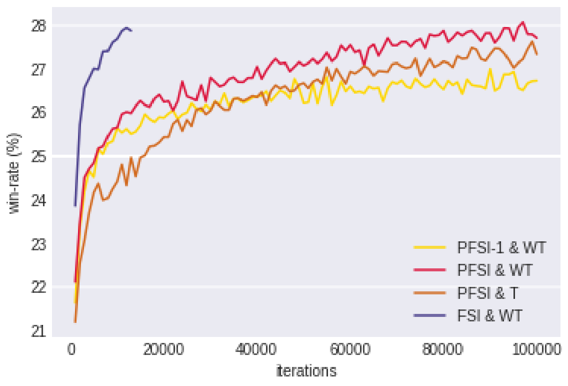

Fig. 7 Evolution of FSI AIs' performance by iteration amount against 3 Heuristic AIs

Table 3 Final win-rates of various AIs (left) against other AIs (top) in 3vs1, 2vs2 and 1vs3 configurations

vs	Random AI		
PPT & WT	**26.29%**	**26.75%**	**27.30%**
MCPT & NM	24.09%	23.09%	21.99%
FSI & WT	**26.37%**	**27.24%**	**28.45%**
PFSI & T	**25.82%**	**26.80%**	**28.27%**
PFSI & WT	**26.12%**	**27.00%**	**27.85%**
vs	Heuristic AI		
PPT & WT	**26.64%**	**27.00%**	**26.78%**
MCPT & NM	23.83%	22.28%	19.35%
FSI & WT	**26.42%**	**27.33%**	**27.81%**
PFSI & T	**26.05%**	**26.77%**	**27.40%**
PFSI & WT	**26.42%**	**27.00%**	**27.85%**
vs	FSI & WT		
PPT & WT	**25.14%**	**25.01%**	24.82%
MCPT & NM	23.00%	20.45%	15.78%
PFSI & T	24.22%	23.36%	22.38%
PFSI & WT	24.61%	24.15%	23.55%
vs	PFSI & WT		
PPT & WT	**25.70%**	**26.21%**	**26.60%**
MCPT & NM	22.49%	19.50%	14.35%
FSI & WT	**25.48%**	**25.85%**	**26.16%**
PFSI & T	24.56%	24.05%	23.52%

In table 3, the AIs used were the ones that were trained longer, ie. the same AIs that were used to determine the last points in each graphics. It should be noted the training time of the different AIs are very different : the PFSIs algorithms were trained for 18 hours, while the PPT, MCPT and FSI were trained for over 60 hours.

5.3 Interpretation

5.3.1 Pass-Through Algorithms

The Monte-Carlo Pass-Through Algorithms have difficulties rivalling the Fixed-Strategy Iterations algorithms. This can be seen clearly it Figure 5, but we could already see symptoms in Figure 4, the couldn't explore new states as fast. The MCPT algorithm with Wait&Type abstraction in particular slows down very quickly and takes a huge amount of training to go over 80%. When retrying MCPT with the maximum abstraction possible "No Memory" – so the minimal amount of abstracted infosets – it was starting to fare better, implying one bottleneck for theses types of AIs are the size of the abstracted infosets to train.

Another problem is that these Pass-Through algorithms are in essence less stable than the other ones. They rely on the randomness of the exploration to be trained in every case, which is less efficient than bulk exploring. When this randomness is superposed to the randomness of Monte-Carlo, we can end up with very low performance, as with what happened to the MCPTs.

The Pruned Pass-Through, although having a lower win-rate against the Heuristic AI, offers surprisingly good results when playing against the other trained AIs: it beats the PFSI player and is on par against FSI (Table 3). PPT does take five times as long as the PFSI, but beating it is nonetheless impressive. This implies that even if the PPT doesn't have a very fully trained all-rounded behaviour, it has a good strategy when playing against other similar players. This is likely due to its greater amount of iterations: the more iteration a CFR agent does, the more it learns to outplay the previous learned strategies. PPT must have a very good strategy for the 95% more likely states it encounters.

5.3.2 Fixed-Strategy Iterations Algorithms

We can see we gained a lot of speed with the PFSI algorithm compared to the FSI one (fig 3). The PFSI-1 is even $\sim$ 500 times faster. If we look at evolution of the win rates per time, we see that the PFSI have a small lead at the beginning, and that the FSI takes 4 times longer to catch up. When we look at the iterations (fig 7), we see that their is a significant loss in efficiency in terms of per iteration, meaning doing full iterations without pruning gives better quality feedback. Even if the PFSI-1 AI trained faster, we see that it actually never gets a win rate higher than the other PFSI, meaning pruning too roughly is counterproductive. The lower win-rate of the PFSI using only the Type abstraction rather than the Wait&Type one is likely caused by the fact that the position of the player relatively to the claimer is a very useful information for making a decision.

A more optimal AI in terms of training speed would likely be an PFSI with dynamic parameters pruning, similarly to [1]. It would likely start off with a very severe pruning, but finish training with a behaviour close to FSI with few pruning.

6. Conclusion

This paper compared the implementations of different CFR-based Artificial Intelligence in 4-player Dudo.

Since the amount of infoset in this environment can't be tackled even by our modern computers, we started by reducing its size by abstracting the information. Using imperfect recall, a method that already proved itself efficient in 2-player Dudo [8], we were able to summarise states by truncating the game history. We then used CFR variants that had been shown working for 2-player Dudo as inspiration to build AIs specifics for 4-player Dudo, either by tweaking their input and parameters, or by combining them.

The experiments confirm that 3 out of the 4 implemented AIs are able to learn how to beat the hand-coded ones, and the AIs introduced in this paper, Pruned Fixed-Strategy Iteration, reaches similar win-rates 4 times faster then the other ones. If we look at the training per iterations, the Fixed-Strategy Iteration (FSI) without pruning proved itself more efficient. The Pass-Through types had in opposite a very bad learning per iteration, but compensate this by speed: the Pruned variation could run an iteration more than 1000 times faster than FSI. When having AIs of different types with similar win-rates compete between on another, we notice the slower AIs – Pruned Pass-Through and Fixed-Strategy Iterations – are better when playing against the fastest PFSI, implying the quality or quantity of the training can more impact against better opponents.

Visualising the difference of performance between the best AIs was difficult, as the win-rates were all tightly around 25%. Future works could envision extending the environment until the actual end of the game (only one player left), or play with more complexes variations of Dudo were round loosers lose dice instead of being directly eliminated. It would also very be interesting to test this AI against human players, to know if these AIs are efficient against human-style game-play.

The AIs are also only using past training, but analysing the strategies of the other players one is opposing is an efficient way to find and exploits weakness in their strategies. We believe further research in CFR applied to Dudo can be accomplished in this direction.

References

[1] Brown, N., Kroer, C. and Sandholm, T.: Dynamic Thresholding and Pruning for Regret Minimization, *AAAI Conference on Artificial Intelligence* (2017).

[2] Brown, N. and Sandholm, T.: Superhuman AI for multiplayer poker, *Science*, (online), DOI: 10.1126/science.aay2400 (2019).

[3] Burch, N., Lanctot, M., Szafron, D. and Gibson, R. G.: Efficient Monte Carlo Counterfactual Regret Minimization in Games with Many Player Actions, *Advances in Neural Information Processing Systems 25* (Pereira, F., Burges, C. J. C., Bottou, L. and Weinberger, K. Q., eds.), Curran Associates, Inc., pp. 1880–1888 (2012).

[4] Chen, C. and Tomoyuki, K.: Counterfactual Regret Minimization for the Board Game Geister, *Game Programming Workshop 2018 Proceedings*, Vol. 2018, pp. 137–144 (2018).

[5] Gibson, R. and Szafron, D.: Regret Minimization in Multiplayer Extensive Games, *International Joint Conference on Artificial Intelligence* (2011).

[6] Laird, J. E., Derbinsky, N. and Tinkerhess, M.: Online determination of value-function structure and action-value estimates for reinforcement learning in a cognitive architecture, *Advances in Cognitive Systems*, Vol. 2, pp. 221–238 (2012).

[7] Lanctot, M., Waugh, K., Zinkevich, M. and Bowling, M.: Monte Carlo Sampling for Regret Minimization in Extensive Games, *Advances in Neural Information Processing Systems 22* (Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I. and Culotta, A., eds.), Curran Associates, Inc., pp. 1078–1086 (2009).

[8] Neller, T. W. and Hnath, S.: Approximating Optimal Dudo Play with Fixed-Strategy Iteration Counterfactual Regret Minimization, *Advances in Computer Games* (van den Herik, H. J. and Plaat, A., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 170–183 (2012).

[9] Risk, N. A. and Szafron, D.: Using Counterfactual Regret Minimization to Create Competitive Multiplayer Poker Agents, *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems, pp. 159–166 (2010).