Recursive Monte Carlo Search for Bridge Card Play

Bruno Bouzy
Nukkai
Paris, France
bbouzy@nukkai.ai

Alexis Rimbaud
Nukkai
Paris, France
arimbaud@nukkai.ai

Véronique Ventos
Nukkai
Paris, France
vventos@nukkai.ai

***Abstract*—Computer Bridge remains a challenging obstacle for Artificial Intelligence. For the last twenty years, the state-of-the-art playing programs have been using a depth-one Monte Carlo (MC) search approach, associated with an open card solver called Double Dummy Solver (DDS). When increasing the computing resources, the MC approach reaches a plateau, and its playing level cannot be improved. In this work, we study Recursive MC (RMC) for Bridge card play. We show that, with more computing resources, this approach performs better than MC. Rather than using DDS or any domain-dependent simulator, a level $N + 1$ RMC consists in using a level N RMC playing program as simulator, level-zero RMC being MC. This recursion mechanism can be iterated several times at the cost of increasing the computing time, each time a recursion level is added. This work focusses on card play with no trump in duplicate with either 13 cards per player or 5 cards per player. With 13 cards per player, level-one RMC is superior to MC with a margin of 0.5 trick per card distribution on average, which is statistically significant. This is the first time RMC is applied with success to computer Bridge card play.**

***Index Terms*—Monte Carlo search, imperfect information games, game of Bridge, card play.**

I. Introduction

After the success of Alpha Zero on complete information games such as Go, Chess and Shogi [1], and after the success of Pluribus on the game of Poker with many players [2], card games with three players or more, such as Skat and Bridge, remain challenging obstacles for artificial intelligence and the computer game community. Computer bridge has been an exciting domain for the last twenty years. The best programs [3] still use the Monte Carlo (MC) approach [4] with the so-called Double Dummy Solver (DDS) as simulator [5]. DDS is a sound and complete solver on open cards. In 1998, Frank and Basin studied the game of Bridge and identified important issues of this approach such as non-locality and strategy fusion [6]. Nowadays, the best programs still use the MC approach with DDS. Non-locality and strategy fusion remain two issues in Computer Bridge.

In its simplest formulation, non-locality means that the value of a given node in the game tree depends not only on the nodes in the sub-tree below this node, but also on a huge number of nodes external to this sub-tree [7], which makes tree search beyond depth two very difficult.

Strategy fusion points out the fact that using a simulator on open cards such as DDS means that this simulator is allowed to use different strategies on different states belonging to the same infoset, where it should not: when playing an incomplete information game, a player actually plays the same action for all states belonging to the same infoset.

This paper describes an experimental research on Recursive Monte Carlo (RMC) for the game of Bridge [8]. As far as we know, our work is the first one studying RMC on the game of Bridge. Our RMC approach is intended to make a first step in two directions: one step toward managing the non-locality issue, and one step toward removing the strategy fusion problem, so as to surpass the current state-of-the-art MC approach. In our work, we compare the results obtained with RMC to the current MC approach. The results are varied. In some conditions, RMC outperforms MC in a statistically significant way. These results are obtained either with DDS or with the random player as simulator. They are obtained on duplicate card play without the bidding phase. We assume that there is no trump. Our results are mainly obtained on regular conditions Bridge card play, i.e. 13 cards per player, or in a simplification of Bridge with 5 cards per player.

Our work is based on two previous works addressing imperfect information games, and more specifically card games. Long and colleagues studied the success of Monte Carlo search in imperfect information games on synthetic trees [7]. Their results conjectured that RMC should work in several card games, including the game of Bridge. Furtak and Buro described the success of recursive Monte Carlo search on synthetic trees and in the game of Skat [8]. In this context, our work shows the actual success of recursive Monte Carlo search in Bridge card play.

The paper is structured as follows. Section II presents Bridge card play. Section III presents the related work on MC on Computer Bridge and on card games. Section IV describes our RMC approach. Section V describes the experiments and our results. Section VI discusses the results. Section VII sums up the current study.

II. Bridge card play

This section explains the card play in the game of Bridge and the card play used in our work. It skips over details not concerning card play directly, i.e. mainly the bidding phase previous to card play. Complete set of rules can be found at [9].

Bridge is a zero-sum game between two pairs of players (4 players in total): South and North in the same pair against West and East in the other one. The name of the player reflects its position around a table. Bridge is played with a deck of 52 cards. A card has a suit and a rank. There are 4 suits (or colors): Spades (♠), Hearts (♡), Diamonds (♢) and Clubs (♣), and 13 ranks (or heights): 2, 3, 4, etc, up to Ten, Jack, Queen, King and Ace, ordered in increasing strength. For example, 4♢ is the four of diamonds. Bridge is a trick-taking card game. The deck is shuffled and dealt with 13 cards per player. A player sees his own cards and not the others' cards. In a first phase, the bidding phase, the players make different bids or pass and the declaring pair that makes the last bid (followed by 3 passes) decide to fulfill the final contract, i.e. they commit to win a certain number of tricks during the second phase. In the second phase, the card play, the pair who won the contract tries to win at least the number of tricks specified by the contract.

The contract specifies a trump (or No Trump (NT)), a number of tricks, and a declarer. The trump corresponds to a suit: Spades, Hearts, Diamonds and Clubs. The trump will be the dominant suit during card play. The number of tricks to win the contract equals the rank of the contract plus 6.

For example, bidding 3 of diamonds means you commit to win 3+6=9 tricks with diamonds as trumps. The declarer is the player from the declaring side who made the first bid in the suit of the final contract. His partner is called the dummy. Dummy will display his cards on the table after the first card (the lead) played by the defending side. Dummy does not participate in card play as the declarer plays his cards and dummy's.

In our work, we neglect the contract and the bidding phase. We assume that there is no trump and no rank. We assume that South is the declarer, North is the dummy, and the West-East pair is the defence.

At the beginning of card play, each player sees his own cards and not the cards of the other players. Card play starts with the player seating on the left of the declarer (West in our work) playing the first card. Then the cards of the dummy are revealed. Then South plays with the cards of the dummy (North). Then East plays, then South plays. When each player has played one card, the strongest card wins the trick.

The suit of the first card of the trick sets the suit requested for the next cards of the trick. The following players are required to follow this suit if possible. Otherwise, they can play any other cards, including a card with a suit corresponding to the trump. The winner of a trick is the one who plays the highest card in the trump suit (if the contract defined a trump suit, or if there is one such card in the trick), otherwise it is the one who plays the highest card in the suit of the trick.

In our work, since we have no trump, the rule is simple: the highest card in the suit of the trick wins the trick. The player who won the current trick is on lead for the next trick. The players play in a clockwise way.

Card play is over when all the cards have been played. With a deck of 52 cards (respectively 20 cards), there are 13 tricks to win (respectively 5 tricks to win).

At the end of a card play with a contract, the declarer-dummy pair wins if the number of tricks won is greater than the number of tricks specified in the contract.

```
$(card, double)$ $MonteCarlo($ $position\ A$, $player\ P$,
  $simulator\ S$, $int\ NCD)$;
begin
  $position\ B1 = A$ ;
  $generate\ SCD\ a\ set\ of\ card\ distributions\ of$
    $size\ NCD$;
  for each legal card C in the hand of P do
    $A$ = $B1$ ;
    $value(C) = 0$ ;
    $play\ C\ on\ A$;
    $position\ B2 = A$ ;
    for each CD in the SCD do
      $A = B2$ ;
      $deal\ CD\ with\ open\ cards\ on\ A$;
      $S\ playout\ A$;
      $value(C) += S\ .\ value(C)$ ;
    end
    $value(C)\ /= NCD$;
  end
  $Cmax = argmax\ value(C)$ ;
  return $(Cmax, value(Cmax))$ ;
end
```

Algorithm 1: The Monte Carlo procedure for card play.

In our work, since we have no contract, the score of one card play is the difference between the number of tricks won by North-South and the number of tricks won by West-East.

Furthermore, to be fair between two pairs A and B, one given deal of cards can be played twice (two card plays): once with pair A playing North-South, and once with pair B, playing North-South. This mechanism is called duplicate play. In our work, we choose to use the duplicate mechanism. The score of a deal played in duplicate is the difference between the score of pair A playing North-South and the score of pair B playing North-South. (Since our player are bots and not human players, they do not memorize the cards of the first card play and they can fairly play the second card play without being suspected of cheating).

We used the duplicate mechanism for fairness purpose but also for a variance reduction purpose as we explain it in Section V.

III. Related work

This section describes how the Monte Carlo approach is used in Computer Bridge and in card games such as Skat.

Computer Bridge [10] research is strongly influenced by the so called Double Dummy Solver, a perfect solver of hands with open cards, an instance of which was developed by Bo Haglund and Soren Hein in the early 2000s [5]. For each card in a hand, DDS gives the number of tricks gained by playing this card under perfect play with open cards. DDS is an alpha-beta algorithm which is publicly available. In our work, we used this instance of DDS.

Under normal play (with hidden cards) standard Bridge playing programs use DDS in the so-called Monte Carlo approach.

Algorithm 1 gives the pseudo-code of this approach. A is the given position. P is the player to play a card. S is the simulator: S is DDS in state-of-the-art Bridge programs. However, S can be any player able to complete a game until the end and return a value. NCD is the number of card distributions used. $B1$ and $B2$ are local positions. SCD is the set of card distributions corresponding to the hidden cards. Its size is NCD. Generally, S $playout$ A means play the game starting on A until the end with simulator S and give the end value (number of future tricks). Specifically, if the simulator S is DDS, it means solve position A with DDS and give the end value under perfect play. The procedure outputs the card to play and its mean value.

In this document, we name this approach the Monte Carlo DDS (MC-D) approach.

In Bridge, MC-D is not surpassed by any other approach so far because of two things. First, although it corresponds to a result on open cards, the DDS value is strongly correlated with winning tricks under hidden cards. Second, it corresponds to a depth-one search and avoids the problem of non-locality arising at depth-two in card game trees and in Imperfect Information Game (IIG) trees in general.

More precisely, non-locality means that the value V of a node X in a IIG tree, where player P is to play, does not depend only on the sub-tree T below X but also on nodes Y *outside* T. Nodes Y intervene in the computation of V because they are situated at a depth greater than two, where players Q different from P have to play, and because they belong to an information set (infoset), built with the viewpoint of a player Q, that also *contains one node included in* T [7]. The size of the set of nodes Y equals the size of infosets of player Q, which can be huge. Consequently, the non-locality issue is as if the branching factor of the IIG tree were multiplied by the size of the infosets.

However, the MC-D approach is criticized by human Bridge players because it suffers from strategy fusion. Because it plays with open cards, for states belonging to the same infoset, DDS may find a state-dependent strategy to compute the value of a card. This behaviour does not respect the principle of playing the same action for all states of the same infoset when playing with hidden cards.

Overall, MC-D is simple, fast, and sufficiently strong to have been the basis of the best computer programs so far, beating alternative approaches [4].

Perfect Information Monte Carlo (PIMC) is another term to name MC-D. It means playing Monte Carlo simulations when the simulator reasons or plays with perfect information. Long and his colleagues studied the success of PIMC on synthetic trees [7]. By extrapolation, their work on synthetic trees shows that PIMC could be appropriate for card games such as Skat, Bridge or Hearts. PIMC is opposed to IIMC (Imperfect Information Monte Carlo). Generally speaking, IIMC is a MC approach in which the simulator plays with hidden cards, i.e. with imperfect information. The IIMC approach does not suffer from strategy fusion.

In Bridge, an approach different from MC-D hardly works. The idea presented in this document is to test a RMC approach for Computer Bridge either with DDS or with a random player. The RMC approach is known in card games in general. Although it uses a perfect information simulator, a PIMC player can be used as a simulator for a IIMC approach. Buro and Furtak assessed recursive PIMC for the game of Skat [8]. They showed that a level-one recursive PIMC with carefully chosen parameters outperforms PIMC in Skat. Our work is very near to this previous work in that our work shows that recursive PIMC outperforms PIMC in Bridge card play.

Recursive MC is a generalization of Nested Monte Carlo search to imperfect information and multi-player games. Nested Monte Carlo search adresses planning problems with one player like solitaire [11]. Bouzy used Nested Monte Carlo Search to solve Weak Schur Number problems [12].

In perfect information games, Monte Carlo Tree Search (MCTS) [13] is the state-of-the-art tree search approach for most of these games. MCTS is famous for its successes on two-player games such as Go. However, it cannot be used without important differences in IIG.

Information Set MCTS (ISMCTS) [14] is an adaptation of MCTS for IIG. In ISMCTS, each node corresponds to an Information Set (infoset) from the viewpoint of a given player. In Single Observer IS MCTS (SO-ISMCTS), IS MCTS develops one tree in which the information sets are from the point of view of the player who is to play at the root. Let us name it P. In Multiple Observer ISMCTS (MO-ISMCTS), ISMCTS develops one tree for each player. In the tree corresponding to player Q, the information sets are built with the viewpoint of Q. In SO-ISMCTS, at depth one, the nodes correspond to IS with P viewpoint although it is it would be correct to have the viewpoint of depth-one player. So, to us, we think that the mean values computed at depth one cannot be accurate.

MO-ISMCTS is a step toward solving this problem. However, the issue of MO-ISMCTS is how to sample correctly the viewpoint of a player Q, given that P cannot see his cards? MO-ISMCTS may sample without respecting the cards of P because Q does not see cards belonging to P. But then the samples are very different from the cards of P, and the algorithm reasons on cards far from reality. This problem is related to the non-locality issue of IIG. Except on toy IIG, MO-ISMCTS appears to be inefficient.

Poker is the most famous IIG for which artificial intelligence is already successful [2]. Counter-Factual Regret (CFR) minimization algorithm is the core algorithm used in computer Poker [15]. Monte Carlo CFR (MCCFR) is the Monte Carlo version of CFR converging to a Nash equilibrium [16]. Online Outcome Sampling (OOS) also converges to a Nash equilibrium [17]. Because Bridge and Poker are both IIG, these algorithms could be in principle applied to Bridge. However, Bridge is a trick-taking game and Poker is not. Moreover, we think that Bridge with 13 cards per player is currently out of the scope of CFR minimization algorithms. Therefore, in our study, we do not consider these algorithms and we focus our effort on RMC search.

IV. Recursive MC

A. Background

The RMC approach consists in using a PIMC or a IIMC as simulator rather than a basic one [8]. In the pseudo-code of Algorithm 1, a level $N+1$ RMC uses a level N RMC as simulator S. When $N=0$, the simulator is a basic one: DDS, the random player, a policy network or any knowledge-based player. We call MC2-D a level 1 RMC using DDS at level 0. We call MC2-R a level 1 RMC using the random player at level 0.

The upside of RMC is that the programming effort is negligible. The downside is that the time used by a recursive IIMC at level $N+1$ in one order of magnitude greater than the recursive IIMC at level N. The goal of this document is to experimentally assess the RMC approach for Computer Bridge.

B. Time complexity

Let us estimate the time complexity of level-one RMC. Let NCD be the number of card distributions used at each recursive level, A the number of actions available and L the length of a game. A is determined by the number of cards of each player. $L=52$ or $L=20$ according to the size of the deck. Let t_{solve} be the time used by DDS to solve an open card problem, and t_{rand} be the time to choose a random move in a hand. Let $tcc(X)$ be the time to choose a card for player X.

We have:

$$tcc(\text{MC-D}) = O(NCD \times A \times t_{\text{solve}}) \quad (1)$$

$$tcc(\text{MC-R}) = O(NCD \times A \times L \times t_{\text{rand}}) \quad (2)$$

Let $tpg(X)$ be the time to play one game for player X. We have:

$$tpg(\text{MC-D}) = O(L \times NCD \times A \times t_{\text{solve}}) \quad (3)$$

$$tpg(\text{MC-R}) = O(L \times NCD \times A \times L \times t_{\text{rand}}) \quad (4)$$

With one recursion, we have:

$$tcc(\text{MC2-D}) = O(NCD^2 \times A^2 \times L \times t_{\text{solve}}) \quad (5)$$

$$tcc(\text{MC2-R}) = O(NCD^2 \times A^2 \times L^2 \times t_{\text{rand}}) \quad (6)$$

Then we have:

$$tpg(\text{MC2-D}) = O(L^2 \times A^2 \times NCD^2 \times t_{\text{solve}}) \quad (7)$$

$$tpg(\text{MC2-R}) = O(L^3 \times A^2 \times NCD^2 \times t_{\text{rand}}) \quad (8)$$

V. Experiments

Globally, the goal of the experiments is to compare Recursive MC against MC with no recursion. The experiments can be performed either with 13 cards per player, or with 5 cards per player. The MC players may use either DDS as simulator, or the random player. The first section explains the settings of the experiments.

Preliminary experiments are necessary to obtain the best MC players, i.e. MC players with no recursion against which the RMC players are confronted. These MC players are described in the second section. A third section describes experiments to assess level-one RMC players. A fourth section describes experiments with level-two RMC. A fifth section describes experiments with level-three RMC.

A. Settings

Playing in duplicate allows a significant variance reduction. In order to compare two pairs of players (A, A) and (B, B), an experiment consists in playing 100 card distributions (CD) played in duplicate. In duplicate, the card play is played twice: once with (A, A) playing North-South, and once with (B, B) playing North-South. The result of the card play in duplicate is the difference between the number of tricks obtained by (A, A) playing North-South and the number of tricks obtained by (B, B) playing North-South. For instance, if the result of (A, A) playing North-South is 10-3 and the result of (B, B) playing North-South is 7-6, then the result of this card distribution in duplicate is +3 for (A, A). The result of an experiment is the average of the results over the 100 CD, which we call μ. When playing with 13 cards per player (resp. 5 cards per player) the standard deviation σ equals 1.2 (resp. 0.5) on average. Therefore, after 100 CD, the standard deviation of the average over the 100 results, which we name $\overline{\sigma}$ in the following, is 0.12 (resp. 0.05) with 13 cards (resp. 5 cards) per player. For a specific experiment, when $\overline{\sigma}$ is different from these values, this will be detailed.

It is very important to use duplicate experiments to reduce variance over card distributions. Without the duplicate mechanism, the scores varies between -13 and +13 and the standard deviation can be between 5 and 10. In duplicate, the scores equal 0, +1 or -1 most of the times, 2 or 3 in absolute value sometimes only, and other values occur very rarely, which yields a standard deviation of roughly 1. Playing in duplicate reduces the standard deviation by a factor of roughly 10. For the same precision on μ, this reduction allows to perform 100 CD instead of 10,000, which is substantial.

For testing, we use two sets of CD, one set for 5 cards per player, and one set for 13 cards per player. These two sets are drawn at random, once for all, and they are the same for all the experiments.

Besides, in this study, we assume that there is no trump.

For scoring, there are several ways to implement a score. The simplest is counting the difference of tricks. When a contract is defined by a bidding phase, two other kinds of scoring are possible: a win-loss scoring giving 1 when the contract has succeeded and 0 otherwise, and a weighted combination of the win-loss scoring with the trick scoring. In this paper, we use the difference of tricks as explained above.

Concerning DDS, we used the instance developed by Bo Haglund and Soren Hein [5].

We used an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz to perform the experiments.

TABLE I
13 CARDS PER PLAYER. GROWING *NCD* EXPERIMENT FOR MC-D. ONE COLUMN FOR EACH CONFRONTATION. A POSITIVE VALUE CORRESPONDS TO A BETTER PLAY FOR THE FIRST PLAYER OF THE COLUMN.

3 vs 1	10 vs 3	30 vs 10	100 vs 30	300 vs 100	1k vs 300
+1.3	+0.8	+0.25	-0.05	+0.10	-0.15

TABLE II
13 CARDS PER PLAYER. GROWING *NCD* EXPERIMENT FOR MC-R AGAINST MC-D-100. ONE COLUMN FOR EACH VALUE OF *NCD*. THE NEGATIVE VALUES MEANS THAT MC-R IS WEAKER THAN MC-D-100.

100	300	1k	3k	10k	30k	100k
-3.9	-3.1	-2.9	-2.7	-2.7	-2.7	-2.8

B. No recursion

The playing level of a MC player depends on the number of card distributions used for the simulations, named NCD in this paper. The greater NCD, the better the playing level. We call MC-X($NCD = y$) the MC player using simulator X and y card distributions. However, for each MC player, a plateau is met when NCD is greater than a specific value, named the plateau value in this study. The plateau value is determined with the growing NCD experiment. For instance, MC-D(NCD=1) plays against MC-D(NCD=3), MC-D(NCD=3) plays against MC-D(NCD=10), and so on.

1) DDS simulator:

a) 13 cards per player: Table I shows the results of the growing experiment on DDS. We see that the plateau value of MC-D equals 100. MC-D-100 is the benchmark in this setting.

b) 5 cards per player: For 5 cards per player with MC-D, the plateau value of NCD equals 20.

2) Random simulator:

a) 13 cards per player: We make MC-R(NCD) play against MC-D-100. Table II show the results. The plateau value of MC-R is roughly 3000. We see that MC-D-100 is 2.7 trick stronger than MC-R-3000 on average.

b) 5 cards per player: For 5 cards per player with MC-R, the plateau value of NCD roughly equals 600.

C. One recursion

When using level-one recursion, two parameters are important: $NCD1$ the number of card distributions dealt at the high level, and $NCD2$ the number of card distributions dealt at the low level. MC2-X-$NCD1$-$NCD2$ names the level one RMC player with X as simulator (X can be D or R). $NCD1$ names the number of card distributions at the high level and $NCD2$ the number of card distributions at the low level. In order to test MC2-X-$NCD1$-$NCD2$ against MC-D-100, we display the results in tables with different values of $NCD1$ and $NCD2$, one line for each value of $NCD2$, one column for each value of $NCD1$. A number in a cell ($NCD1$, $NCD2$) corresponds to μ, the result of one experiment (100 CD played in duplicate).

1) Random simulator:

a) 13 cards per player: We assessed MC2-R-$NCD1$-$NCD2$. Globally, MC2-R-$NCD1$-$NCD2$ outperforms MC-R-$NCD1$ and MC-R-$NCD2$. For low values of $NCD1$ and $NCD2$, the greater $NCD1$ and $NCD2$, the better the playing level. We assessed MC2-R-$NCD1$-$NCD2$ against MC-D-100. Table III shows the results.

Overall and roughly, we observe that the greater $NCD1$, the better MC2-R-$NCD1$-$NCD2$. So far, the best results are obtained with $NCD1 \times NCD2 = 300k$. The average score is $+0.10$ with MC2-R-3k-100, and $+0.08$ with MC2-R-30k-10. These 2 positive results are promising (after 100 CD, the standard deviation equals 0.10). With $NCD1 \times NCD2 = 1M$ the results are slightly negative. These results make us think that MC2-R can be on a par only with MC-D-100 but can hardly surpass it. Beside, MC2-R-300-30 is a good compromise between quality of play and time used: average score equals -0.4 and 10 minutes are used for playing one CD in duplicate.

TABLE III
13 CARDS PER PLAYER. MC2-R-*NCD1*-*NCD2* VS MC-D-100. ONE LINE FOR EACH VALUE OF *NCD2*. ONE COLUMN FOR EACH VALUE OF *NCD1*. MEAN VALUES OVER 100 CD. $\overline{\sigma} = 0.10$.

	30	100	300	1k	3k	10k	30k
10	-2.9	-1.3	-0.9	-1.0	-0.70	-0.22	**+0.08**
30	-2.3	-1.2	-0.4	-0.30	-0.10	-0.04	-0.17
100	-0.9	-0.6	-0.54	-0.14	**+0.10**	-0.02	
300	-1.1	-0.22	-0.34	-0.16			

b) 5 cards per player: We assessed MC2-R-$NCD1$-$NCD2$ against MC-D-20. Table IV shows the results.

TABLE IV
5 CARDS PER PLAYER. MC2-R-*NCD1*-*NCD2* VS MC-D-20. ONE LINE FOR EACH VALUE OF *NCD2*. ONE COLUMN FOR EACH VALUE OF *NCD1*. MEAN VALUES OVER 100 CD. $\overline{\sigma} = 0.05$.

	10	20	25	30	40	60	100
10	-0.3	-0.24	-0.12	-0.16			
20	-0.22	-0.10	-0.14	-0.12			
25	-0.34	0.00	+0.04	**+0.10**	-0.24		
30	-0.34	+0.04	-0.02	+0.03	-0.12	-0.32	
40	-0.40	-0.08	**+0.10**	+0.04	-0.14	-0.20	
60	-0.24	+0.06	+0.02	+0.02	-0.16	-0.24	-0.30
100	-0.34	0.00	0.00	+0.06	-0.06	-0.26	-0.24

Obtaining this table took about 5 minutes to get one result in a cell. We observe that $20 \leq NCD1 \leq 30$ and $25 \leq NCD2$ gives the best results. MC2-R(25, 40) and MC2-R(30, 25) give the best results: $+0.10$. The standard deviation is roughly 0.05 after 100 CD. Therefore, with the 2 sigma rule (with 95% confidence), this result is statistically significant.

2) DDS simulator:

a) 13 cards per player: This paragraph assesses MC2-D-$NCD1$-$NCD2$ against MC-D-100 with different values of $NCD1$ and $NCD2$ with 13 cards per player. Table V shows the results. Today, the two best results are $+0.57$ by MC2-D-300-30 and $+0.58$ by MC2-D-300-100. Because the standard deviation is 0.14 after 100 CD, saying that MC2-D-300-30 and MC2-D-300-100 are superior to MC-D-100 is statistically

TABLE V

13 CARDS PER PLAYER. MC2-D-*NCD1*-*NCD2* VS MC-D-100. ONE LINE FOR EACH VALUE OF *NCD2*. ONE COLUMN FOR EACH VALUE OF *NCD1*. MEAN VALUES OVER 100 CD. $\overline{\sigma} = 0.15$.

	3	10	30	100	300	1000
1	-7.0	-2.9	-1.4	-1.1	-0.56	-0.23
3	-2.8	-1.9	-0.40	**+0.19**	**+0.18**	+0.03
10	-2.9	-1.06	-0.12	**+0.31**	**+0.44**	**+0.41**
30	-2.0	-0.14	-0.12	**+0.52**	**+0.57**	
100	-2.3	0.00	+0.11	**+0.36**	**+0.58**	

TABLE VI

5 CARDS PER PLAYER. MC2-D-*NCD1*-*NCD2* VS MC-D-20. ONE LINE FOR EACH VALUE OF *NCD2*. ONE COLUMN FOR EACH VALUE OF *NCD1*. MEAN VALUES OVER 100 CD. $\overline{\sigma} = 0.05$.

	1	3	10	20	30	60
1	-1.6	-1.1				
3	-1.7	-0.9	-0.26	-0.02	-0.08	-0.28
10		-0.74	-0.18	**+0.16**	+0.06	-0.26
20			-0.16	**+0.24**	+0.02	-0.06
30			-0.32	+0.06	+0.06	-0.22
60			-0.12	+0.10	-0.04	-0.24

significant. One CD of the (100, 100) experiment lasts 12 hours.

b) 5 cards per player: This paragraph assesses MC2-D-*NCD1*-*NCD2* against MC-D-20 with different values of *NCD1* and *NCD2* with 5 cards per player. Table VI shows the results.

Today, the two best results are $+0.24$ by MC2-D-20-20 and $+0.16$ by MC2-D-20-10. Because the standard deviation is 0.05 after 100 CD, these two results show that MC2-D-20-10 and MC2-D-20-20 are superior to MC-D-20 with statistical significance. 100 CD of the (20, 20) (resp. (60, 60)) experiment lasts about 1 hour (resp. 10 hours).

D. Two recursions

In this section, we aim at assessing level-two RMC. With 52 cards, our experiments lasted a long time, which forbids testing high values of *NCD* for the 3 levels of MC3-R or MC3-D. So as to observe the behaviour level-two recursive MC more quickly, we assess MC3-R only and not MC3-D, and we launch experiments with 5 cards per player. In this setting, MC-D-20 is our benchmark, and we assess MC3-R against it.

1) Random simulator:

a) 5 cards per player: This paragraph assesses MC3-R-*NCD*-*NCD*-*NCD* against MC-D-20 with different values of *NCD* with 5 cards per player. Table VII shows the results. (To simplify the complexity of these experiments, *NCD* is common to all recursion levels).

Today, the two best results are $+0.20$ by MC3-R-25-25-25 and $+0.14$ by MC3-R-30-30-30. Because the standard deviation is 0.05 after 100 CD, these two results are significantly positive. 100 CD of the (20, 20, 20) (resp. (30, 30, 30)) experiment lasts about 2 hours (resp. 5 hours). We observed that increasing *NCD* above 40 was not beneficial.

TABLE VII

5 CARDS PER PLAYER. MC3-R-*NCD*-*NCD*-*NCD* VS MC-D-20. ONE COLUMN FOR EACH VALUE OF *NCD*. MEAN VALUES OVER 100 CD. $\overline{\sigma} = 0.05$.

10	16	18	20	25	30	32	36
-0.34	-0.38	+0.04	+0.08	**+0.20**	**+0.14**	+0.10	-0.06

TABLE VIII

5 CARDS PER PLAYER. MC4-R-*NCD*-*NCD*-*NCD*-*NCD* VS MC-D-20. ONE COLUMN FOR EACH VALUE OF *NCD*. MEAN VALUES OVER 100 CD. $\overline{\sigma} = 0.05$.

1	5	10	15	20	25	30
-1.8	-0.34	-0.18	-0.19	**+0.16**	+0.02	0.00

E. Three recursions

In this section, we assess MC4-R, and not MC4-D, in experiments with 5 cards per player. MC-D-20 is still our benchmark.

1) Random simulator:

a) 5 cards per player: This paragraph assesses MC4-R-*NCD*-*NCD*-*NCD*-*NCD* against MC-D-20 with different values of *NCD* with 5 cards per player. Table VIII shows the results. (To simplify the complexity of these experiments, *NCD* is common to all recursion levels).

Today, the best result is $+0.16$ by MC4-R-20-20-20-20. The standard deviation is 0.05 after 100 CD. 100 CD of the (20, 20, 20, 20) experiment lasts about 4 days. Increasing *NCD* above 30 was not tried. Because this result is far from being superior to the result obtained with two recursions, this experiment shows that the third recursion is actually not beneficial.

VI. DISCUSSION

This section discusses the results, firstly, in the normal game with 13 cards per player, then, in a simplification with 5 cards per player.

A. 13 cards per player

1) Novelty: Within the game of Bridge, the work is very novel in that for the last twenty years, MC-D remains the state-of-the-art. For the first time, MC-D is surpassed very significantly by MC2-D. Always, in the domain of Bridge, a second novelty is that MC2-R is on a par with MC-D, which means that DDS is not necessary to achieve the level of the state of the art. This result opens up the path toward learning approaches without knowledge, and at least without DDS.

Within card games, the first time that recursive MC beat MC was for the game of Skat [8]. However, Skat is less complex than Bridge: 35 cards in Skat versus 52 in Bridge, 3 players in Skat versus 4 players in Bridge, 9 tricks in Skat versus 13 in Bridge. For this reason, our result is novel.

2) Significance: Our results show that with sufficient computing time, MC-D can be surpassed in a statistically significant way by RMC-D. After 100 CD, we have $\overline{\sigma} = 0.15$. The average difference between RMC-D and MC-D equals $+0.58$.

Thus, the 3-sigma rule says that RMC-D is superior to MC-D with probability 0.99.

Furthermore, in Bridge standards, +0.5 trick per card distribution on average in duplicate is huge. Intuitively, it means that, for each deal (card distribution), either RMC-D wins 1 trick more than MC-D, or RMC-D wins the same number of tricks. When you think that a Bridge contract can be won or lost for 1 trick, it may mean that RMC-D may win 50% of contracts and may draws the other 50% of deals. Half-a-trick difference on average is huge.

Because the tests are performed in duplicate, another way to show the significance of the result is to count the number of deals in which RMC-D wins more tricks than MC-D (a win), or the same number of tricks (a draw), or less tricks than MC-D (a loss), i.e. the number of wins, draws and losses. Over 100 deals, the $+0.58$ result corresponds to 49 wins for RMC-D, 35 draws and 16 losses, which is striking.

3) Zero-knowledge approaches are possible: With 13 cards per player, our work shows that the random player is a serious option to be used as a simulator. First, a level-one RMC approach using the random player as simulator is far better than the simple MC approach using the random player as simulator (more than $+2$ tricks per CD). Second, a level-one RMC approach using the random player as simulator plays on a par with the current MC-D approach (without surpassing it). This result is very interesting because it is obtained without knowledge. It means that other zero-knowledge approaches can be fruitful as well, in particular any learning approach like Expert Iteration (ExIt) [18].

4) Limitation: With 13 cards per player, the time to perform one deal by some of our best players, MC2-D-100-30 or MC2-R-10k-30, is roughly 5 hours, which is currently impractical in the context of real play against human players. However, MC2-D-100-3, playing one deal in less than 30' can be a good compromise between playing level $(+0.19)$ and computing time. Optimizing the computing time of our players more accurately is an interesting perspective.

B. 5 cards per player

With 5 cards per player, we performed RMC experiments as well. With MC2-D (resp. with MC2-R), we obtain $+0.24$ (resp. $+0.10$) against MC-D-20. We performed a level-2 (resp. level-3) recursion experiment with MC3-R (resp. MC4-R), and we obtain $+0.20$ (resp. $+0.16$) against MC-D-20. These experiments confirm the results obtained with 13 cards per player: MC-D is significantly surpassed by MC2-D and MC2-R slightly surpasses MC-D (or, at least, MC2-R plays on par with MC-D). However in our experiments, growing the number of recursions is not effective: against MC-D, MC3-R and MC4-R obtain results not better than those obtained by MC2-R.

Furthermore, with 5 cards per player, we observe that the rule G saying that "the greater the NCD, the better the playing level" seems to be wrong for RMC. Our tables of results obtained with 5 cards per player show that the best results are obtained when $NCD1$ ranges in a specific interval: $[20, 30]$. For higher values of $NCD1$ the results are worse. This observation is hard to explain. We think that RMC may amplify anomalies specific to simple MC. Rule G is perhaps wrong with simple MC as well. We want to investigate on this issue in a future work.

VII. Conclusion

In this research, we studied Recursive Monte Carlo Search for Bridge card play. We obtained positive and significant results in some conditions. With 13 cards per player against MC-D-100, the most significant result is $+0.5$ by MC2-D. This result experimentally shows that level-one recursion is useful to make a first step to break the non-locality issue of Bridge. Beside, still with 13 cards per player against MC-D-100, MC2-R obtains $+0.1$, showing that, within a one-level RMC framework, the random simulator, i.e. a zero-knowledge approach, is a possible alternative to DDS as simulator.

With 5 cards per player, we performed level-one recursion experiments with the random simulator and with DDS. With MC2-D (resp. with MC2-R), we obtained $+0.24$ (resp. $+0.10$) against MC-D-20. Furthermore, with 5 cards per player, we performed a level-2 (resp. level-3) recursion experiment with MC3-R (resp. MC4-R), and we obtained $+0.20$ (resp. $+0.16$). These results confirm the good results obtained with 13 cards per player. However, they also show that the recursion mechanism does not add up well.

Because the zero-knowledge approach MC2-R works, the future work may consist in approximating MC-R or MC-D with a learning neural network which can be used as simulator. However, approximating MC-R or MC-D can be difficult [19]. With such an approximation, we hope to obtain results as good as those obtained with MC2-R ou MC2-D so far. Then, if this works well, we want to iterate this approach as done in Expert Iteration (ExIt) [18] or in AlphaStar [1].

References

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm," *arxiv*, 2017.

[2] N. Brown and T. Sandholm, "Superhuman AI for multiplayer Poker," *Science*, vol. 365, no. 6456, pp. 885–890, August 2019.

[3] Y. Costel, "Wbridge5," http://www.wbridge5.com/.

[4] M. L. Ginsberg, "GIB: Steps toward an expert-level bridge-playing program," in *IJCAI*, 1999, pp. 584–589.

[5] B. Haglund and S. Hein, "Double dummy solver," https://github.com/dds-bridge/dds.

[6] I. Frank and D. Basin, "Search in games with incomplete information: a case study using Bridge card play," *Artificial Intelligence Journal*, vol. 100, pp. 87–123, 1998.

[7] J. Long, N. Sturtevant, M. Buro, and T. Furtak, "Understanding the success of perfect information Monte Carlo sampling in game tree search," in *AAAI*, 2010, pp. 134–140.

[8] T. Furtak and M. Buro, "Recursive Monte Carlo search for imperfect information games," in *CIG-2013*, 2013, pp. 1–8.

[9] A. C. B. League, "How to play Bridge," www.acbl.org.

[10] V. Ventos and O. Teytaud, "Le Bridge, nouveau défi de l' intelligence artificielle ?" *Revue d'Intelligence Artificielle*, vol. 31, no. 3, pp. 249–279, 2017.

[11] T. Cazenave, "Nested Monte Carlo search," in *IJCAI*, 2009, pp. 456–461.

[12] B. Bouzy, "An abstract procedure to compute weak Schur number lower bounds," LIPADE, Paris Descartes University, Tech. Rep., 2015.

[13] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte-Carlo Tree Search methods," *IEEE TCIAIG*, vol. 4, no. 1, pp. 1–43, 2012.

[14] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information Set Monte Carlo Tree Search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120–143, 2012. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6203567

[15] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," in *NIPS*, 2007, pp. 1729–1736.

[16] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling, "Monte Carlo sampling for regret minimization in extensive games," in *NIPS*, 2009, pp. 1078–1086.

[17] V. Lisý, M. Lanctot, and M. Bowling, "Online Monte Carlo counterfactual regret minimization for search in imperfect information games," in *AAMAS*, 2015, pp. 21–34.

[18] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," in *NIPS*, 2017, pp. 5366–5376.

[19] K. Mossakowski and J. Mandciuk, "Learning without human expertise. a case study of double dummy Bridge problem," *IEEE Transactions on Neural Networks*, vol. 20, no. 2, pp. 278–299, February 2009.