The $\alpha\mu$ Search Algorithm for the Game of Bridge

Tristan Cazenave[1] and Véronique Ventos[2]

[1] LAMSADE, Université Paris-Dauphine, PSL, CNRS, France
Tristan.Cazenave@dauphine.psl.eu
[2] NUKKAI, Paris, France vventos@nukk.ai

Abstract. $\alpha\mu$ is an anytime heuristic search algorithm for incomplete information games that assumes perfect information for the opponents. $\alpha\mu$ addresses and if given enough time solves the strategy fusion and the non-locality problems encountered by Perfect Information Monte Carlo search (PIMC). Strategy fusion is due to PIMC playing different strategies in different worlds when it has to find a unique strategy for all the worlds. Non-locality is due to choosing locally optimal moves that are globally inferior. In this paper $\alpha\mu$ is applied to the game of Bridge and outperforms PIMC.

1 Introduction

As computer programs have reached superhuman at Go [18] and other two-player perfect information games like Chess and Shogi [17] starting from zero knowledge, some of the next challenges in games are imperfect information games such as Bridge or Poker. Multiplayer Poker has been solved very recently [1] while Computer Bridge programs are still not superhuman.

The state of the art for Computer Bridge is Perfect Information Monte Carlo search. It is a popular algorithm for imperfect information games. It was first proposed by Levy [12] for Bridge, and used in the popular program GIB [9]. PIMC can be used in other trick-taking card games such as Skat [2,11], Spades and Hearts [20]. The best Bridge and Skat programs use PIMC. Long analyzed the reasons why PIMC is successful in these games [14].

However PIMC plays sub-optimally due to two main problems: strategy fusion and non-locality. We will illustrate these problems in the second section. Frank and Basin [5] have proposed a heuristic algorithm to solve Bridge endgames that addresses the problems of strategy fusion and non-locality for late endgames. The algorithm we propose is an improvement over the algorithm of Frank and Basin since it solves exactly the endgames instead of heuristically and since it can also be used in any state even if the search would be too time consuming for the program to reach terminal states. Ginsberg has proposed to use a lattice and binary decision diagrams to improve the approach of Frank and Basin for solving Bridge endgames [9]. He states that he was generally able to solve 32 cards endings, but that the running times were increasing by two orders of magnitude as each additional card was added. $\alpha\mu$ is also able to solve Bridge endings but it can also give a heuristic answer at any time and for any number of cards and adding cards or searching deeper does not increase as much the running time.

Furtak has proposed recursive Monte Carlo search for Skat [7] to improve on PIMC but the algorithm does not give exact results in the endgame and does not solve the non-locality problem.

Other approaches to imperfect information games are Information Set Monte Carlo Tree Search [3], counterfactual regret minimization [22], and Exploitability Descent [13].

$\alpha\mu$ searches with partial orders. It is related to partial order bounding [16] and to opponent modeling in card games [19]. However our algorithm is different from these algorithms since it searches over vectors only composed of 0 and 1 and uses different backups for sets of vectors at Max and Min nodes as well as probabilities of winning.

The contributions of the paper are:

1. An anytime heuristic search algorithm that assumes Min players have perfect information and that improves on PIMC and previous related search algorithms.
2. An anytime solution to the strategy fusion problem of PIMC that solves the strategy fusion problem when given enough time.
3. An anytime solution to the non-locality problem of PIMC using Pareto fronts of vectors representing the outcomes for the different possible worlds. It also converges given enough time.
4. A search algorithm with Pareto fronts.
5. The description of the early and root cuts that speed up the search.
6. Adaptation of a transposition table to the algorithm so as to improve the search speed using iterative deepening.
7. Experimental results for the game of Bridge.

The paper is organized as follows: the second section deals with Bridge and Computer Bridge. The third section defines vectors of outcomes and Pareto fronts. The fourth section details the $\alpha\mu$ algorithm. The fifth section gives experimental results.

2 Bridge and Computer Bridge

2.1 Bridge in Short

The interested reader can refer for instance to [15] for a more complete presentation of the game of Bridge. Bridge is a trick-taking card game opposing four players (denoted by West, North, East and South or W,N,E,S) divided in two partnerships (East-West and North-South). A standard 52 card pack is shuffled and each player receives a hand of 13 cards that is only visible to him. A Bridge game is divided into two major playing phases: the bidding phase (out of the scope of the paper) and the card play. The goal of the bidding phase is to reach a contract which determines the minimum number of tricks the pair commits to win during the card play, either with no trump (NT) or with a determined suit as trump. During the card play, the goal is to fulfill (for the declarer) or to defeat (for the defenders) the contract reached during the bidding phase. Let us assume that South is the agent who plays the game (i.e the declarer). Player on the left of the declarer (W) exposes the first card of the game. The declarer's partner (N called Dummy) then lays his cards face up on the table. When playing in a NT contract, there is only one simple rule : each player is required to follow suit if possible and can play

any card of the suit. When the four players have played a card, the player who played the highest-ranked card in the suit (2<3<...<10<J<Q<K<A) wins the trick and he will be on lead at the following trick. The game is over when all the cards have been played, In the following (including in our experiments), we assume that the pair North-South (NS) reached the contract of 3NT. In this case, if NS wins nine or more tricks the game is won otherwise it is lost [3].

2.2 Computer Bridge

A Double Dummy Solver (DDS) is a solver for complete information Bridge. A very efficient Double Dummy Solver (DDS) has been written by Bo Haglund [10]. In our experiments we use it to evaluate double dummy hands. It makes use of partition search [8] among many other optimizations to improve the solving speed of the $\alpha\beta$.

PIMC is the state of the art of Computer Bridge, it is used for example in GIB [9] and in WBRIDGE5 [21] the former computer world champion.

The PIMC algorithm is given in algorithm 1. In this algorithm S is the set of possible worlds and $allMoves$ is the set of moves to be evaluated. The play function plays a move in a possible world and returns the corresponding state. All the possible worlds have the same hand for the player to play, so all the moves for the player to play are legal in all the possible worlds. The DDS function evaluates the state using a double dummy solver. The DDS sends back the maximum number of tricks the player to play can win. If the number of tricks already won by the declarer plus the number of tricks that the declarer can win returned by DDS is greater than or equal to the contract the world is evaluated to 1, else to 0.

Algorithm 1 The PIMC algorithm.

```
1: Function PIMC (allMoves, S)
2:   for move ∈ allMoves do
3:     score[move] ← 0
4:     for w ∈ S do
5:       s ← play (move, w)
6:       score[move] ← score[move]+ DDS (s)
7:     end for
8:   end for
9:   return argmax_move(score[move])
```

PIMC accumulates the payoff of strategies related to different worlds. This process leads to an optimistic evaluation since in reality a specific strategy has to be chosen. This problem is known as strategy fusion [4]. The reason why PIMC is optimistic is that it can adapt its strategy to each world since it has perfect information. In the real game the player does not know the real world and cannot adapt its strategy, it has to choose a strategy working in all possible worlds.

[3] It is an acceptable simplification of the real scoring of Bridge. At Bridge, the declarer has to make six more tricks than the number in his contract.

Fig. 1: Example of a Bridge hand illustrating strategy fusion.

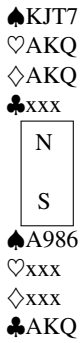
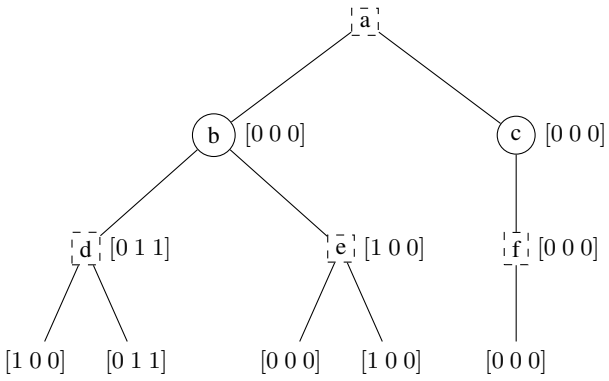


Figure 1 is a hand from [9] which illustrates the strategy fusion problem. PIMC finds that the declarer always makes all of the four tricks at Spades when it has only 50% chances of making them since it has to finesse the Queen. The declarer does not know where is the Queen, so it has to bet where she is and for example play the Jack for the dummy hoping she is not in East hand.

Strategy fusion arises because PIMC can play different cards in different worlds whereas it should play the same cards in all the worlds since it cannot distinguish between worlds. Frank and Basin solve this problem with an algorithm they call Vector Minimaxing [5] that plays the same cards for the Max player in all the worlds.

Fig. 2: Example of a tree with three worlds illustrating non-locality.



The fact that a move is optimal at a node but not optimal considering the whole search tree leads to the problem of non-locality. From an algorithmic point of view non-locality can be explained using figure 2 from [6]. It illustrates non-locality when searching with strategy fusion for Max and perfect information for Min. As usual the Max nodes are squares and the Min nodes are circles. The Max nodes are dashed since

they represent information sets for Max. Max does not know in which of the three worlds he is playing whereas Min can distinguish the worlds and thus chooses actions in a different way for each world. The leaves give the result of the game in the three possible worlds. For example the move to the left from node d reaches a state labeled $[1\ 0\ 0]$ which means that the game is won in world 1 (hence the 1 in the first position), lost in world 2 (hence the 0 in the second position) and also lost in world 3 (hence the 0 in the third position). The vectors near the internal nodes give the values that are backed up by the strategy fusion for Max and perfect information for Min algorithm. We can see that each Max node is evaluated by choosing the move that gives the maximum average outcome. For example at node d there are two moves, the left one leads to $[1\ 0\ 0]$ and therefore has an average of $\frac{1}{3}$ whereas the right one leads to $[0\ 1\ 1]$ and has an average of $\frac{2}{3}$. So node d backs up $[0\ 1\ 1]$. However it is not globally optimal. If instead of choosing the right move at node d it chooses the left move it backs up $[1\ 0\ 0]$ and then the b node would have been evaluated better also with $[1\ 0\ 0]$. It illustrates that choosing the local optimum at node d prevents from finding the real optimum at node b. At Min nodes the algorithm chooses for each world the minimal outcome over all children since it can choose the move it prefers most in each different world.

3 Vectors of Outcomes and Pareto Fronts

In this section we define Vectors and Pareto fronts that are used by the algorithms in the next section.

3.1 Definitions for Vectors

Given n different possible worlds, a vector of size n keeps the status of the game for each possible world. A zero (resp. one) at index i means that the game is lost (resp. won) for world number i.

Associated to each vector there is another vector of booleans indicating which worlds among the n are possible in the current state. At the root of the search all worlds are possible but when an opponent makes a move, the move is usually only valid in some of the worlds, the associated vector is then updated by changing from true to false for these worlds.

The associated vector is used to define the domination between two vectors:

v1 $\geq$ v2 iff $\forall i \in [1, n]$, v1[i] $\geq$ v2[i]
v1 dominates v2 iff they have the same associated worlds,
v1 $\geq$ v2 and $\exists i \in [1, n]$ such that v1[i] $>$ v2[i].

The score of a vector is the average among all possible worlds of the values contained in the vector.

3.2 Pareto Front

A Pareto front is a set of vectors. It maintains the set of vectors that are not dominated by other vectors. When a new vector is a candidate for insertion in the front the first

thing to verify is whether the candidate vector is dominated by a vector in the front or equal to another vector in the front. If it is the case the candidate vector is not inserted and the front stays the same. If the candidate vector is not dominated it is inserted in the front and all the vectors in the front that are dominated by the candidate vector are removed.

For example consider the Pareto front $\{[1\ 0\ 0], [0\ 1\ 1]\}$. If the vector $[0\ 0\ 1]$ is a candidate for entering the front, then the front stays unchanged since $[0\ 0\ 1]$ is dominated by $[0\ 1\ 1]$. If we add the vector $[1\ 1\ 0]$ then the vector $[1\ 0\ 0]$ is removed from the front since it is dominated by $[1\ 1\ 0]$, and then $[1\ 1\ 0]$ is inserted in the front. The new front becomes $\{[1\ 1\ 0], [0\ 1\ 1]\}$.

It is useful to compare Pareto fronts. A Pareto front P_1 dominates or is equal to a Pareto front P_2 iff $\forall v \in P_2$, $\exists v' \in P_1$ such that (v' dominates v) or v'=v.

4 The $\alpha\mu$ Algorithm

In this section we first explain how the algorithm deals with strategy fusion and non-locality, we then give some optimizations and we eventually explain the details of the search algorithm.

4.1 Search with Strategy Fusion

Let us assume that the defense knows the cards of the declarer and that the declarer optimizes against all possible states that correspond to his information. The score of a move for the declarer is the highest score of all vectors in the Pareto front of the move. At a Max node the declarer computes after each move the union of the Pareto fronts of all the moves that have been tried so far. Min has knowledge of the declarer cards so in each world he takes the move that minimizes the result of Max. The code for Min and Max nodes is given in algorithm 2. $\alpha\mu$ is a generalization of PIMC since a search with a depth of one is PIMC.

The parameter M controls the number of Max moves, when $M = 0$ the algorithm reaches a leaf and each remaining possible world is evaluated with a double dummy search. The stop function also stops the search if the game is already won no matter what is played after. The parameter $state$ contains the current state where all the moves before have been played and which does not contain the hidden information. The parameter $Worlds$ contains the set of all possible worlds compatible with the moves already played. The transposition table contains the Pareto front and the best move found by the previous search of a state. If the state has not yet been searched the Pareto front is initialized as the empty set and the best move is not set. If at Min node, the set of all possible moves in all possible worlds is calculated (lines 13-16). At each played move the list of possible worlds is updated and a recursive call performed. The Pareto front resulting from the recursive call is then combined with the overall front (lines 18-23). We will explain later the min algorithm. Similar operations are performed for a Max node except that the combination with the overall front is then done with the max algorithm (lines 27-45). We explain the max algorithm in the next subsection. The optimizations and detailed explanations of the algorithm are given in subsections 4.3 and 4.4.

Algorithm 2 The $\alpha\mu$ search algorithm with cuts and transposition table.

```
1: Function αμ (state, M, Worlds, α)
2:   if stop(state, M, Worlds, result) then
3:     update the transposition table
4:     return result
5:   end if
6:   t ← entry of state in the transposition table
7:   if Min node then
8:     mini ← ∅
9:     if t.front ≤ α then
10:      return mini
11:    end if
12:    allMoves ← ∅
13:    for w ∈ Worlds do
14:      l ← legalMoves (w)
15:      allMoves = allMoves ∪ l
16:    end for
17:    move t.move in front of allMoves
18:    for move ∈ allMoves do
19:      s ← play (move, state)
20:      W1 ← {w ∈ Worlds : move ∈ w}
21:      f ← αμ (s, M, W1, ∅)
22:      mini ← min(mini, f)
23:    end for
24:    update the transposition table
25:    return mini
26:  else
27:    front ← ∅
28:    allMoves ← ∅
29:    for w ∈ Worlds do
30:      l ← legalMoves (w)
31:      allMoves = allMoves ∪ l
32:    end for
33:    move t.move in front of allMoves
34:    for move ∈ allMoves do
35:      s ← play (move, state)
36:      W1 ← {w ∈ Worlds : move ∈ w}
37:      f ← αμ (s, M − 1, W1, front)
38:      front ← max(front, f)
39:      if root node then
40:        if μ(front) = μ of previous search then
41:          break
42:        end if
43:      end if
44:    end for
45:    update the transposition table
46:    return front
47:  end if
```

4.2 Dealing with Non-locality

Max nodes At Max nodes each possible move returns a Pareto front. The overall Pareto front is the union of all the Pareto fronts of the moves. The idea is to keep all the possible options for Max, i.e. Max has the choice between all the vectors of the overall Pareto front.

Min nodes The Min players can choose different moves in different possible worlds. So they take the minimum outcome over all the possible moves for a possible world. So when they can choose between two vectors they take for each index the minimum between the two values at this index of the two vectors.

Now when Min moves lead to Pareto fronts, the Max player can choose any member of the Pareto front. For two possible moves of Min, the Max player can also choose any combination of a vector in the Pareto front of the first move and of a vector in the Pareto front of the second move. In order to build the Pareto front at a Min node we therefore have to compute all the combinations of the vectors in the Pareto fronts of all the Min moves. For each combination the minimum outcome is kept so as to produce a unique vector. Then this vector is inserted in the Pareto front of the Min node.

An example of the product of Pareto fronts is given in figure 3. We can see in the figure that the left move for Min at node a leads to a Max node b with two moves. The Pareto front of this Max node is the union of the two vectors at the leaves: $\{[0\ 1\ 1], [1\ 1\ 0]\}$. The right move for Min leads to a Max node c with three possible moves. When adding the vectors to the Pareto front of the Max node c, the algorithm sees that $[1\ 0\ 0]$ is dominated by $[1\ 0\ 1]$ and therefore does not add it to the Pareto front at node c. So the resulting Pareto front for the Max node c is $\{[1\ 1\ 0], [1\ 0\ 1]\}$. Now to compute the Pareto front for the root Min node we perform the product of the two reduced Pareto fronts of the children Max nodes and it gives: $\{[0\ 1\ 0], [0\ 0\ 1], [1\ 1\ 0], [1\ 0\ 0]\}$. We then reduce the Pareto front of the Min node and remove $[0\ 1\ 0]$ which is dominated by $[1\ 1\ 0]$ and also remove $[1\ 0\ 0]$ which is also dominated by $[1\ 1\ 0]$. Therefore the resulting Pareto front for the root Min node is $\{[0\ 0\ 1], [1\ 1\ 0]\}$.

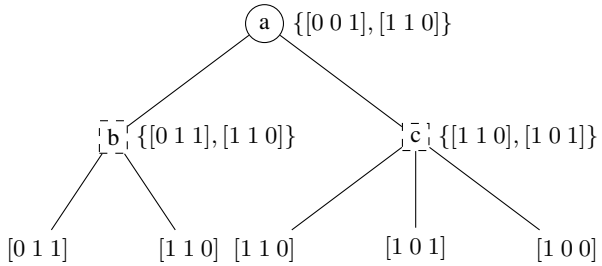
We can also explain the behavior at Min nodes on the non-locality example of figure 2. The Pareto front at Max node d is $\{[1\ 0\ 0], [0\ 1\ 1]\}$. The Pareto front at Max node e is $\{[0\ 0\ 0], [1\ 0\ 0]\}$. It is reduced to $\{[1\ 0\ 0]\}$ since $[0\ 0\ 0]$ is dominated. Now at node b the product of the Pareto fronts at nodes d and e gives $\{[1\ 0\ 0], [0\ 0\ 0]\}$ which is also reduced to $\{[1\ 0\ 0]\}$. The Max player can now see that the b node is better than the c node, it was not the case for the strategy fusion algorithm without Pareto fronts.

4.3 Optimizations

In this section we explain how to speedup search.

Skipping Min nodes A one ply search at a Min node will always give the same result as the Pareto front at that node since the Double Dummy Solver has already searched all worlds with an $\alpha\beta$. The Min player can choose the move for each world and therefore will have the same result as the $\alpha\beta$ for each world.

Fig. 3: Product of Pareto fronts at Min nodes.



This is why we only keep the number M of Max moves to be played in the search. The search will never stop after a Min move since recursive calls at Min node do not decrease M. This is intended since the results of the search after a Min move are the same as before the Min move.

Iterative Deepening and Transposition Table Iterative deepening starts with one Max move and increases the number of Max moves at every iteration. The number of Max moves is the number of Max nodes that have been traversed before reaching the current state. The results of previous searches for all the nodes searched are stored in a transposition table.

An entry in the transposition table contains the Pareto front of the previous search at this node and the best move found by the search. The best move is the move with the greatest probability.

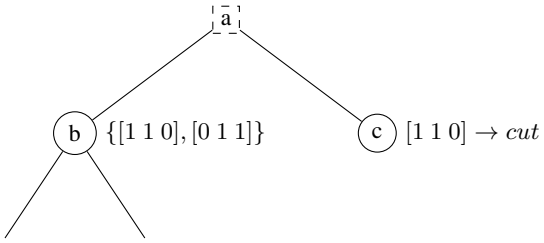
When a search is finished at a node, the entry in the transposition table for this node is updated with the new Pareto front and the new best move.

Comparing Pareto Fronts at Min Nodes When a Pareto front P_1 dominates another Pareto front P_2 it is safe to ignore the move associated to P_2 since it adds no options to P_1. If it is true for the current front P_2 at a Min node it will also be true when searching more this Min node since P_2 can only be reduced to an even more dominated Pareto front by more search at a Min node.

The Early Cut If a Pareto front at a Min node is dominated by the Pareto front of the upper Max node it can safely be cut since the evaluation is optimistic for the Max player. The Max player cannot get a better evaluation by searching more under the Min node and it will always be cut whatever the search below the node returns since the search below will return a Pareto front smaller or equal to the current Pareto front. It comes from the observation that a world lost at a node can never become won.

Figure 4 gives an example of an early cut at a Min node. The root node a is a Max node, the first move played at a returned $\{[1\ 1\ 0], [0\ 1\ 1]\}$ which is backed up at node a. The second move is then tried leading to node c and the initial Pareto front calculated with double dummy searches at node c is [1 1 0]. It is dominated by the Pareto front of node a so node c can be cut.

Fig. 4: Example of an early cut at node c.



The Root Cut If a move at the root of $\alpha\mu$ for M Max moves gives the same probability of winning than the best move of the previous iteration of iterative deepening for $M-1$ Max moves, the search can safely be stopped since it is not possible to find a better move. A deeper search will always return a worse probability than the previous search because of strategy fusion. Therefore if the probability is equal to the one of the best move of the previous shallower search the probability cannot be improved and a better move cannot be found so it is safe to cut.

4.4 Detailed Algorithm

$\alpha\mu$ with transposition table and cuts is a search algorithm using Pareto fronts as evaluations and bounds. The algorithm is given in algorithm 2.

The evaluation of a state at a leaf node is the double dummy evaluation for each possible world. An evaluation for a world is 0 if the game is lost for the Max player and 1 if the game is won for the Max player (lines 2-5).

The algorithm starts with getting the entry t of $state$ in the transposition table (line 6). The entry contains the last Pareto front found for this state and the best move found for this state, i.e. the move associated to the best average.

If the state is associated to a Min node, i.e. a Min player is to play, the algorithm starts to get the previously calculated Pareto front from the transposition table (line 8). Then it looks for an early cut (lines 9-11). If the node is not cut it computes the set of all possible moves over all the valid worlds (lines 12-16). It then moves the move of the transposition table in front of the possible moves (line 17). After that it tries all possible moves (line 18). For each possible move it computes the set W_1 of worlds still valid after the move and recursively calls $\alpha\mu$ (lines 19-21) . The parameters of the recursive call are s, the current state, M the number of Max moves to go which is unchanged since we just played a Min move, W_1 the set of valid worlds after $move$, and an empty set for $alpha$ to avoid deeper cuts. The front returned by the recursive call is then combined to the current front using the min function (line 22). When the search is finished it updates the transposition table and returns the $mini$ Pareto front (lines 24-25).

If the state is associated to a Max node it initializes the resulting front with an empty set (line 27). Then as in the Min nodes it computes the set of all possible moves and moves the transposition table move in front of all the possible moves (lines 28-32). Then it tries all the moves and for each move computes the new set W_1 of valid worlds

and recursively calls $\alpha\mu$ with $M-1$ since a Max move has just been played and $front$ as $alpha$ since a cut can happen below when the move does not improve $front$ (lines 33-36). The resulting front f is combined with front with the max function (line 37). If the score of the best move ($\mu(front)$) is equal to the score of the best move of the previous search and the node is the root node then a Root cut is performed (lines 38-42). When the search is finished the transposition table is updated and $front$ is returned (lines 44-45).

The search with strategy fusion is always more difficult for the Max player than the double dummy search where the Max player can choose different moves in the different possible worlds for the same state. Therefore if a double dummy search returns a loss in a possible world, it is sure that the search with $\alpha\mu$ will also return a loss for this world.

If the search is performed until terminal nodes and all possible worlds are considered then $\alpha\mu$ solves the strategy fusion and the non locality problems for the game where the defense has perfect information.

If the search is stopped before terminal nodes and not all possible worlds are considered then $\alpha\mu$ is a heuristic search algorithm. The algorithm is named $\alpha\mu$ since it maximizes the mean and uses an α bound.

5 Experimental Results

In our experiments we fix the bid so as to concentrate on the evaluation of the card play. We use duplicate scoring. It means that the different evaluated programs will play the same initial deals against the same opponents. When $\alpha\mu$ is the declarer it will play against two PIMC as the defense. In the following we note $\alpha\mu(1)$ for $\alpha\mu$ with one Max move and $\alpha\mu(3)$ for $\alpha\mu$ with three Max moves. In order to compare $\alpha\mu$ as a declarer to PIMC as a declarer we compare $\alpha\mu(3)$ as a declarer to $\alpha\mu(1)$ as a declarer since $\alpha\mu(1)$ is PIMC. In most of the experiments $\alpha\mu$ with 20 possible worlds plays against PIMC with 20 possible worlds as the defense. All results are computed playing the same initial deals with the same seed for each deal, meaning that as long as the card played are the same, the generated possible worlds for $\alpha\mu(1)$ and $\alpha\mu(3)$ are the same. The Pareto fronts stay small in practice. For Bridge it is very often the case with random deals that a world champion and a weak player score the same. It is more informative to compute statistics on deals with different results. Challenges in Bridge are played with carefully selected deals not random ones.

Table 1 gives the results for games that have a different result for $\alpha\mu(1)$ and $\alpha\mu(3)$. We only keep the games that are either won for $\alpha\mu(1)$ and lost for $\alpha\mu(3)$ or lost for $\alpha\mu(1)$ and won for $\alpha\mu(3)$. We stop the experiment when 200 such games have been played. The winrate of $\alpha\mu(3)$ is significantly greater than $\alpha\mu(1)$. For example with 32 cards, $\alpha\mu(3)$ wins 62% of the time with a standard deviation of 3.4%.

PIMC (i.e. $\alpha\mu(1)$) is already a very strong player as a declarer so improving on it even slightly is difficult. We can conclude that looking three Max moves ahead is beneficial and that $\alpha\mu(3)$ improves on PIMC.

In practice the number of vectors in the Pareto fronts always stayed very small in the games it played.

Table 1: Comparison of $\alpha\mu(3)$ with PIMC for games that have a different result. 3NT contract.

Cards	M	Worlds	Games	Winrate	σ
32	1	20	200	0.380	0.034
32	3	20	200	0.620	0.034
36	1	20	200	0.455	0.035
36	3	20	200	0.545	0.035
40	1	20	200	0.450	0.035
40	3	20	200	0.550	0.035
52	1	20	200	0.420	0.035
52	3	20	200	0.580	0.035

Table 2 gives the percentage of games played by $\alpha\mu(3)$ that are different from the games played by $\alpha\mu(1)$ it also gives the percentage of the total number of games that have different results. In practice when generating the initial deals randomly, many deals are not interesting: they are either easily won or completely lost. The two algorithms play the same cards on these deals as there is no interesting alternative. This is the reason why we only keep the games where the algorithms have different results so as to evaluate the difference between the two algorithms.

Table 2: Percentage of different games for $\alpha\mu(3)$ for the 3NT contract.

Cards	Worlds	Different	Different Results
32	20	10.4%	2.4%
36	20	16.4%	3.8%
40	20	23.6%	5.8%
52	20	40.9%	10.6%

We now compare the times to play moves with and without Transposition Tables and cuts. Table 3 gives the average time per move of different configurations of $\alpha\mu$ playing entire games. The initial deals used for this experiment are the first 100 initial deals of the previous experiment for a total of 2 600 cards played. TT means Transposition Table, R means Root Cut, E means Early Cut. We can observe that a Transposition Table associated to cuts improves the search time. For $M = 1$ the search time is 0.159 seconds. For $M = 3$ without transposition table and cuts the average search time per move is 55 seconds. When using a transposition table associated to early and root cuts it goes down to 3 seconds.

We also made experiments with the 7NT contract. In this contract the declarer has to win all the tricks. When a trick is won by the defense the search can stop as the contract is lost. Table 4 gives the results for 2, 3 and 4 Max moves and 20 and 40 possible worlds. We compute statistics on the games that have different results for PIMC and $\alpha\mu$ out of the 10 000 games played for each experiment. For example the first line gives the comparison of $\alpha\mu$ with two Max moves and 20 worlds against PIMC with 20

Table 3: Comparison of the average time per move of different configurations of $\alpha\mu$ on deals with 52 cards for the 3NT contract.

Cards	M	Worlds	TT	R	E	Time
52	1	20				0.118
52	2	20	n	n	n	1.054
52	2	20	y	y	n	0.512
52	2	20	y	n	y	0.503
52	2	20	y	y	y	0.433
52	3	20	n	n	n	10.276
52	3	20	y	y	n	3.891
52	3	20	y	n	y	1.950
52	3	20	y	y	y	1.176

worlds for 10 000 games, 283 of these 10 000 games give different outcomes for the two algorithms, 64.3% of these 283 games are won by $\alpha\mu$ and lost by PIMC. For the 40 worlds experiments $\alpha\mu$ with 40 worlds is compared to PIMC with 40 worlds.

Table 4: Comparison of $\alpha\mu$ versus PIMC for the 7NT contract, playing 10 000 games.

Cards	M	Worlds	$\neq$ results	Winrate	σ
52	2	20	283	0.643	0.0285
52	3	20	333	0.673	0.0257
52	4	20	374	0.679	0.0241
52	2	40	324	0.630	0.0268
52	3	40	347	0.637	0.0258
52	4	40	368	0.655	0.0248

6 Conclusion and Future Work

We presented $\alpha\mu$, which is a heuristic search algorithm for incomplete information games. In order to highlight its advantages, we tested $\alpha\mu$ on the card play of Bridge, which is known to be difficult for classical search algorithms such as PIMC according to the strategy fusion and non-locality problems. To solve the non-locality problem $\alpha\mu$ uses Pareto fronts as evaluations of states and combines them in an original way at Min and Max nodes. To solve the strategy fusion problem it plays the same moves in all the valid worlds during search. Experimental results for the 3NT contract and even more for the 7NT contract show it significantly improves on PIMC, which is a breakthrough in the field of Computer Bridge.

We also presented the use of a transposition table as well as the early and the root cut for $\alpha\mu$. When searching three Max moves ahead it enables the search to be faster while returning the same move as the longer search without the optimizations.

In future work we expect to use partition Search with $\alpha\mu$ and to speed it up with other cuts and optimizations. We plan to carry out new experiments with $\alpha\mu$ for the defense, and with real scores instead of only win/loss. We also plan to parallelize the algorithm. The algorithm is easy to parallelize strongly, for example parallelizing the DDS calls at the leaves or parallelizing the search for the different moves at the root. The sequential times are not indicative of the time limits once parallelized. Finally, a promising approach for improving $\alpha\mu$ is to make inferences linked to the strategy of the opponents in order to reduce the set of possible worlds.

Acknowledgment

Thanks to Alexis Rimbaud for explaining me how to use the solver of Bo Haglund and to Bo Haglund for his Double Dummy Solver.

References

1. Brown, N., Sandholm, T.: Superhuman AI for multiplayer poker. Science **365**(6456), 885–890 (2019)
2. Buro, M., Long, J.R., Furtak, T., Sturtevant, N.: Improving state evaluation, inference, and search in trick-based card games. In: Twenty-First International Joint Conference on Artificial Intelligence (2009)
3. Cowling, P.I., Powley, E.J., Whitehouse, D.: Information set monte carlo tree search. IEEE Transactions on Computational Intelligence and AI in Games **4**(2), 120–143 (2012)
4. Frank, I., Basin, D.: Search in games with incomplete information: A case study using bridge card play. Artificial Intelligence **100**(1-2), 87–123 (1998)
5. Frank, I., Basin, D.: A theoretical and empirical investigation of search in imperfect information games. Theoretical Computer Science **252**(1-2), 217–256 (2001)
6. Frank, I., Basin, D.A., Matsubara, H.: Finding optimal strategies for imperfect information games. In: AAAI/IAAI. pp. 500–507 (1998)
7. Furtak, T., Buro, M.: Recursive monte carlo search for imperfect information games. In: 2013 IEEE Conference on Computational Inteligence in Games (CIG). pp. 1–8. IEEE (2013)
8. Ginsberg, M.L.: Partition search. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 1. pp. 228–233 (1996), http://www.aaai.org/Library/AAAI/1996/aaai96-034.php
9. Ginsberg, M.L.: GIB: imperfect information in a computationally challenging game. J. Artif. Intell. Res. **14**, 303–358 (2001). https://doi.org/10.1613/jair.820, https://doi.org/10.1613/jair.820
10. Haglund, B.: Search algorithms for a bridge double dummy solver. Tech. rep. (2010)
11. Kupferschmid, S., Helmert, M.: A skat player based on monte-carlo simulation. In: International Conference on Computers and Games. pp. 135–147. Springer (2006)
12. Levy, D.N.: The million pound bridge program. Heuristic Programming in Artificial Intelligence The First Computer Olympiad pp. 95–103 (1989)
13. Lockhart, E., Lanctot, M., Pérolat, J., Lespiau, J.B., Morrill, D., Timbers, F., Tuyls, K.: Computing approximate equilibria in sequential adversarial games by exploitability descent. arXiv preprint arXiv:1903.05614 (2019)

14. Long, J.R., Sturtevant, N.R., Buro, M., Furtak, T.: Understanding the success of perfect information monte carlo sampling in game tree search. In: Twenty-Fourth AAAI Conference on Artificial Intelligence (2010)
15. Mahmood, Z., A., G., Sharif, O.: Bridge for Beginners: A Complete Course. Pavilion Books (2014)
16. Müller, M.: Partial order bounding: A new approach to evaluation in game tree search. Artificial Intelligence **129**(1-2), 279–311 (2001)
17. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
18. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of go without human knowledge. Nature **550**(7676), 354 (2017)
19. Sturtevant, N., Zinkevich, M., Bowling, M.: Prob-maxˆ n: Playing n-player games with opponent models. In: AAAI. vol. 6, pp. 1057–1063 (2006)
20. Sturtevant, N.R., White, A.M.: Feature construction for reinforcement learning in hearts. In: International Conference on Computers and Games. pp. 122–134. Springer (2006)
21. Ventos, V., Costel, Y., Teytaud, O., Ventos, S.T.: Boosting a bridge artificial intelligence. In: 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI). pp. 1280–1287. IEEE (2017)
22. Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. In: Advances in neural information processing systems. pp. 1729–1736 (2008)