Moss Side Whist

20916873 Evan Huang

Introduction

Trick-taking card games, as imperfect information, stochastic, sequential competitive multiplayer games, represent both a serious challenge for researchers in the field of artificial intelligence as well as an opportunity to develop novel techniques. At the same time, suitability of trick-playing games as a domain is derived from the fixed maximum search depths, as cards are sequentially discarded a set amount of times in a set amount of tricks, (Bethe 2010) and the lessened emphasis on deceptive play compared with other forms of imperfect information card games, such as poker (Russell & Norvig 2016).

Attempts by researchers to develop Game-playing Artificial intelligence (AI) implementations in such games (contract bridge, spades, whist variations, etc.) have had varied levels of success over the last 20 years.

Literature Review

Earlier efforts dating back to the 1980s involved theory crafting around knowledge-based systems. For instance, Yegnanarayana *et al.* (1996) theorised of using an artificial neural network to capture the reasoning processed used by humans when bidding in bridge. Smith *et. al* (1998), attempted to mimic human knowledge-based reasoning through HTN (Hierchical Task Network Planning), abstracting a set of problems down to simpler forms.

The development of a HTN framework led to success at the World Computer Bridge Championship in 1997 with an agent, *Bridge Baron*, based on this approach. However, *Bridge Baron* was one of the few successful game-playing systems to use hierarchical planning (Russell & Norvig, 2016) and was still unable to compete with expert level human players (Palma, 2014), or deliver consistent good amateur-level performance (Bethe 2010; Ginsberg 2002).

Ginsberg (2002) comments on the shift in methodology after 1998 from human-imitating, knowledge-based agents (Frank, 1998) towards a brute-force search approach. Ginsberg (2002) developed a program, Ginsberg's intelligent Bridge-player (*GIB*), which utilised Monte Carlo sampling of card orderings to plan moves. Ginsberg used an explanation-based generalization approach that stored rules of excellent moves and generalizations in different standard situations of the game of Bridge. (Palma 2014) with play described as expert level coming 12th in a tournament of bridge with 35 human players without bidding. (Ginsberg 2002; Benthe 2010).

More recent developments in search-based techniques is the use of Monte Carlo Tree search, first theorised in 2006 by Remi Coulom, (Palma, 2014) with the Upper Confidence Bounds for Tree algorithm (UCT) formalized by Kocsis & Szepesvari (2006). While Ginsberg calculates the probability of every state, a Monte Carlo Tree Searching algorithm simply simulates down through the tree instead of calculating probability. Monte Carlo Tree Search [MCTS] based search techniques has led to successful outcomes in games such as Go (Lee *et al.*, 2010), regardless of its large search space complexities. In a notable recent development, Cowling *et al.* (2014) used an Information Set Monte Carlo Tree Search, a modified variant of MCTS where the nodes of the tree represented information sets, to find an optimal solution with 2500 iterations in a quarter of a second on an Android smartphone, demonstrating the feasibility of "brute-force" search-based techniques.

Monte Carlo Tree Search

Palma (2014) makes the case for Monte Carlo tree searching over other search techniques for playing card games in 3 main aspects. Firstly, Palma (2014) notes that the algorithm is aheuristic, that is, it does not need to know of any existing strategies or techniques for the game to be effective, and can be easily applied to any domain which can be modelled as a tree. It is sufficient to simply model the ruleset of the game for the algorithm to run. (Russell & Norvig, 2016)

Secondly, the algorithm is anytime, Palma (2014) notes that it can be terminated at any stage of its search, and immediately return a result. This is useful given the hard computational limit imposed on agents in the project, and allows the algorithm to run for the maximum possible number of iterations, refining the best possible estimate of the best move before returning a result.

Third, it is asymmetric, allowing the algorithm to explore the most promising parts of the search space and allowing for an asymmetric growth of the tree, "adapting to the topology of the search space" (Di Palma, 2014).

Additionally, in trick-playing games such as Bridge, with a fair and uniform dealing procedure, where most of the uncertainty is derived from the original distribution of the deal and its randomness, and not from a deceptive or adversarial play (although these still do exist). (Russell & Norvig 2016), a trait which is exploited by a Monte Carlo distribution.

Finally, the framework of Monte Carlo Tree Search is flexible in the ability to adjust expansion policy, reward values and thus the algorithm has a degree of modularity. For example, Cowling et. al (2014), incorporate knowledge-based elements into a Monte Carlo Tree Search implementation to create more believable play for human opponents whilst Chaslot et. al (2010) incorporated Opponent modelling in their poker-playing agent through the use of a Bayesian Classifier.

Game Analysis

Moss Side Whist is related to Bridge within the family of trick-taking, imperfect information, zero-sum stochastic card games, involving 3 players. Like other games of this category, the lack of perfect information precludes any use of a minimax, alpha-beta pruning or A* heuristic strategy, as there is an element of uncertainty about the opponents' state and thus optimal moves. The unique variation with the rules of Moss Side Whist is the removal of one player compared to standard 4-player trick-taking games, as well as the incorporation of a discarding stage.

In just the playing stage, the game has a large maximum branching factor (32), but a fixed depth, 48, as the three players discard one card from their hands of 16 cards over a fixed number of 16 tricks. The player is certain about the cards in his own hand, but initially has no knowledge of the cards in the other players' hands.

The worst case for this game occurs in the situation where the in every round, every player has the ability to play every card in his hand. This can occur for instance when the leader has all 13 trumps and the 7 highest honour cards of another suit, (following the logic laid out by Frank (1998)) and is guaranteed to win and lead every trick, whilst the opposing players are always forced to play a card out of suit, ensuring that every card in their hand represents a valid move in the game tree. Furthermore, the leader has the ability to discard 4 cards of his hand, which he can do 20**C**4 ways (4845).

Moss Side Whist thus has an upper bound on its search space complexity of 16!32! for the playing stage of the game, and the total search space complexity has an upper bound of 20**C**4 x 16!32! ≈ 4845 x 5.50 x 10^{48} ≈ 2.67 x 10^{52}. By comparison, the upper bound of the search space complexity of chess is approximately 10^{47}. A double-blind game of Bridge (perfect information) has a search space complexity of 10^{52}.

An estimation on the lower bound of the number of possible game states can be inferred by only examining the choices that the current player has to make and ignoring opponent choices (Frank 1998). 16! = 2.09 X 10^{13}

In terms of the game tree, a fully-expanded game tree would have a size of 16!32! leaf nodes, or 5.51 x 10^{48}. Using Palma (2014)'s equations the total number of nodes is 6.05 x 10^{49}. Using the EBF Equation from Russell & Norvig (2016), the Effective Branching Factor (EBF) is approximately ≈11.40

$$\text{TotalNodes} = 1 + \sum_{i=1}^{16} \sum_{j=1}^{3} \prod_{x=i}^{16} x \prod_{y=3(i-1)+j}^{32} y$$

$$treeNodes = \frac{EBF^{treeDepth+1} - 1}{EBF - 1},\ EBF > 0.$$

(Equations adapted from Palma, 2014)

Thus the information sets Monte Carlo implementation of a game-playing agent has a similar complexity to Nine Men's Morris in terms of its tree size, and a branching factor similar to Fanorona or Nine Men's Morris, and somewhat significantly higher than normal, 4-player variants of trick-playing games.

As noted by Cowling et. al (2014) the ISMUCT algorithm, whilst performing well at the playing stage of the game, is not an ideal choice for any bidding (in this case, discarding) step as it would extend the depth of the tree and be unlikely to produce a very good outcome. Instead both a random as well as a greedy, rule-based strategy of always discarding the lowest suit was done instead.

Implementation – Tree Search

My implementation of MCTS leans heavily on the framework or skeleton set out by Palma (2014).

All Monte Carlo Tree Based Agents rely on there being an estimated/expected reward of an action that can be estimated through repeatedly running random simulations. (Palma, 2014). These rewards over time from visiting the least visited nodes to the highest scoring nodes. (Palma, 2014).

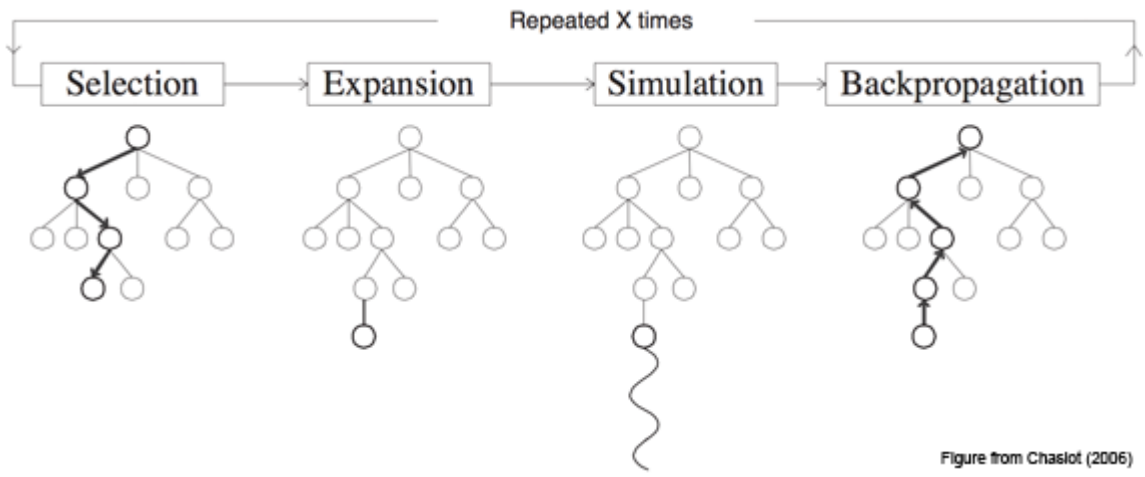


Figure from Chaslot (2006)

The agent thus conducts a form of reinforcement learning (similar to a Bellman-style correction over time), in that the game tree is slowly built up and the agent is initially allowed to explore its

environment, and in subsequent iterations uses the reward information it has gained from those first few iterations to expand the tree in a certain direction based on past experience. MCTS Algorithms thus expand only the most promising areas of the tree. (Palma 2014), and as previously mentioned can be terminated at any stage and return a result given limited computational resources. The algorithm has 4 parts to its operation, incorporating a utility evaluation function (or tree policy), an expansion or rollout function (default policy), a backpropagation reward distributing function, which all operate in turn in an inner loop.

Before initialising the tree, the agent first "guesses" the composition of the opponent's deck by taking the current game state, cloning it, and randomly distributing the cards to each player. This represents one determinization of the information set, and the algorithm is specifically resolving the current (random) case.

Beginning from the root node, the algorithm, in the selection phase, first selects and moves through nodes according to a utility function, (in this case Kocsis & Szepesvari (2006)'s upper confidence bounds for trees). This function balances between selecting of nodes of high value but also prioritising less explored nodes. When a node is encountered that has not had its neighbours added to the tree yet, the selection phase stops and expands a neighbouring node.

The algorithm then runs a rollout policy, which selects according to the default (usually a random distribution) policy until a terminal state is reached. When this happens, the final back propagating stage (for this loop) gets the reward for the terminal state, and sets the reward for all the previously visited parent nodes to this value. Once the loop repeats, the algorithm creates a new determinization by once again cloning and randomising opponent hands (in its simulation of the game state).

The utility function chosen, the upper confidence bound for information set trees, or IS-UCT, initially created by Kocsis & Szepesvari (2006) as the upper confidence bound (USB) function to solve the multi-armed bandit problem, has been applied successfully by Palma (2014), Cowling *et. al* (2014), Chaslot *et al.* (2010) and others. The original function faces an exploitation-exploration dilemma (Palma 2014), which is resolved by the modified function.

$$ISUCT(n') = \frac{Q(n')}{N(n')} + 2C_p\sqrt{\frac{\ln N'(n')}{N(n')}}$$

For example, when N is zero for the child node, and thus this node has never been picked before, the returned value for this is infinity, and thus this node will be selected.

Pseudocode for Information Sets Monte Carlo Tree Search

```
1  procedure ISMCTS_Search(IS0)
2      create root node n0 with Information Set IS0
3      while within computational budget do
4          randomly choose determinization ∈ IS0 -> d0
5          TreePolicy(n1, d0) -> (n1, d1)
6          DefaultPolicy(d1) -> Δ
7          BackPropogate(n1, Δ)
8      end while
9      return a(BestChild(n0)
10 end procedure
11
12 procedure TreePolicy(n,d)
13     while d is nonterminal do
14         if (n,d) has unexplored children then
15             return Expand(n,d)
16         else
17             for all n' (children of n) ∈ movesMade(n,d) do
18                 n'.searched +1 -> n'.searched
19              end for
20              BestISUCTChild(n,d,c) -> n
21              doMove(d, n) -> d
22         end if
23     end while
24     return n
25 end procedure
26
27 procedure Expand(n, d)
28      randomly choose a move ∈ unexplored(n,d) -> m
29      add a new child n' to n
30      doMove(d,m) -> d
31      m -> n'.action
32      return (n', d)
33 end procedure
34
35 procedure BESTISUCTChild(n, d, c)
36      return max (child (n') ∈ movesMade(n,d)
37          according to Q/N + c √(2 ln N' / N)
38 end procedure
39
40 procedure DefaultPolicy (d)
41     while d is nonterminal do
42         choose uniformly at random from validMoves(d) -> a
43         doMove(d,a) -> d
44     end while
45     return reward for state d
46 end procedure
47
48 procedure BackPropogate (n, Δ)
49     while n is not null do
50         n.N+1 -> n.N
51         n.Q+1 -> n.Q
52         parent of n -> n
53     end while
54 end procedure
```

```
 1 procedure discardLowSuit(hand)
 2     for each card > JACK in hand do
 3         case based on suit
 4             case S : card -> spadesList
 5             case D : card -> diamondsList
 6             case C : card -> cardsList
 7             case H : card -> heartsList
 8             end case
 9     end for
10     for each Suit != Trump do
11        [Suit, <suit>List.size()]-> PriorityQ
12     end for
13     [TRUMPSUIT, MAX_VALUE] -> PriorityQ
14     while discard.size < 4 do
15         PriorityQ.poll() -> tm
16     end while
17
18     for i = 1 to 4 do
19         first element of tmp -> discard
20     end for
21     return discard
22 end procedure
```

The chosen discard strategy (Discard Lowest Suit or DLS) was to discard the 4 lowest, non-honour, non-trump cards in the lowest suit or suits, to increase the chances of ruffing a suit and winning a trick.

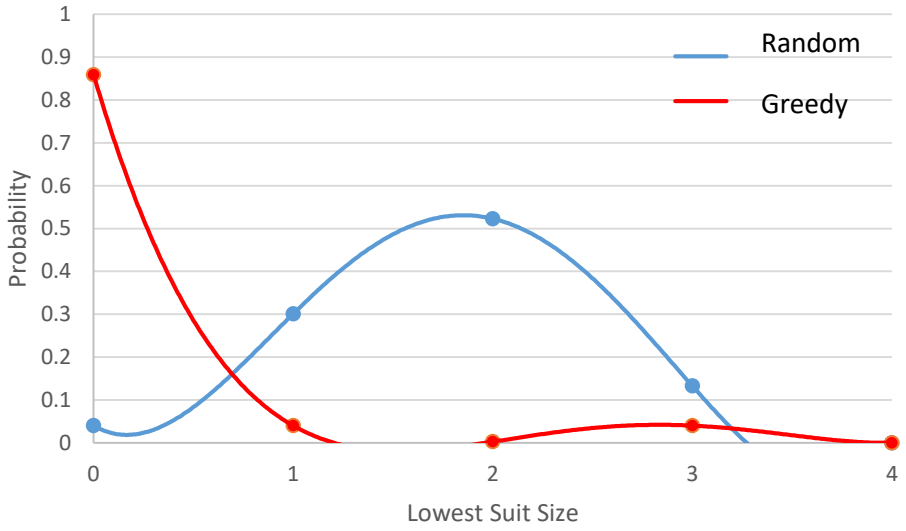
```
 1 procedure emptySuitEvaluator(hand)
 2     for each Suit
 3         if exhausted = true then
 4             for each card = Suit in hand do
 5                     card -> exhausted
 6             end for
 7             if exhausted.size = 0 then
 8                break
 9             end if
10             if notLeader && discard.size < 4 then
11                use learned knowledge to set -> remove
12                for i = 1 to remove do
13                     use discard modelling to determine -> card
14                     exhausted.remove(card)
15                 end for
16              end if
17              getExhaustedPlayer() -> addTo
18              exhausted -> addTo.hand
19          end if
20      end for
```

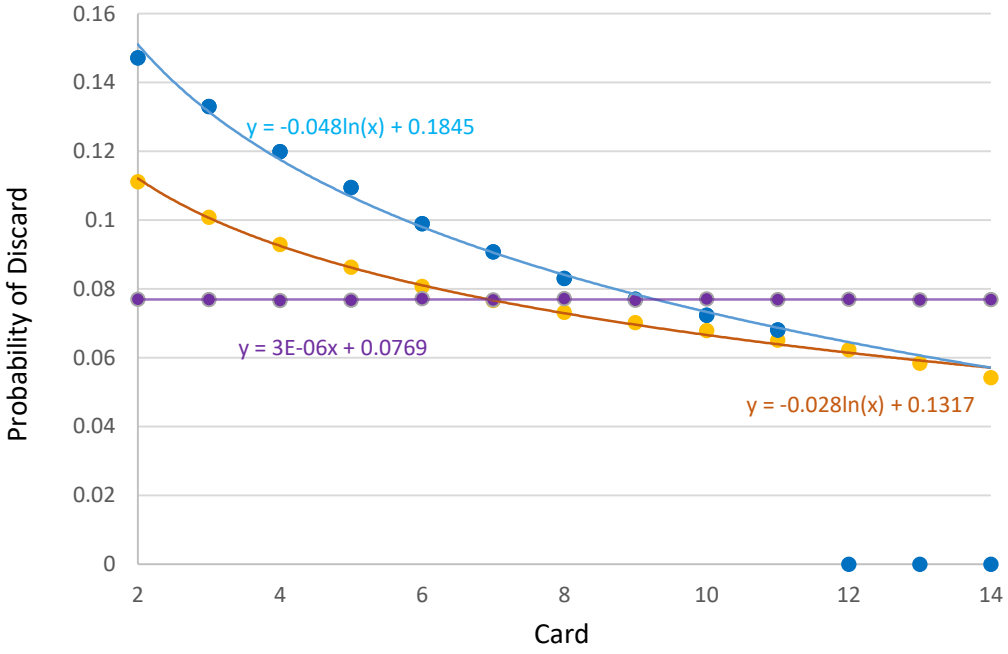
The use of an empty suit evaluating (ESE) function is borne out of, firstly, a need to reduce the state space to a more manageable level, and the observation that when an opposing player runs out of a suit and plays a card out of suit, due to the fact that there are only 2 opposing players, that the remaining opponent must have all the cards that are not A) in the player's hand B) already played in previous tricks and C) were not discarded at the beginning of the round. Using a card-counting strategy, the player is thus able to guess with a high degree of certainty where the remaining card in that exhausted suit are and how they are distributed.

This is especially true if the player was the leader of the current round, and thus has no uncertainty over which cards were discarded at the beginning of the round, and thus can place all the remaining cards of that suit in the other opponent's hands.

The evaluator also ensures that the search is not exploring non-valid game states (where opponent players are playing a suit that he has already exhausted).

In order to determine the exact parameters of the distributions of discarded hands given a greedy discarding policy, as well as a random one, a test class was written to deal 100,000 hands using the same dealing policy as coded in the game, and returning the amount of cards in the smallest suit in the hand.





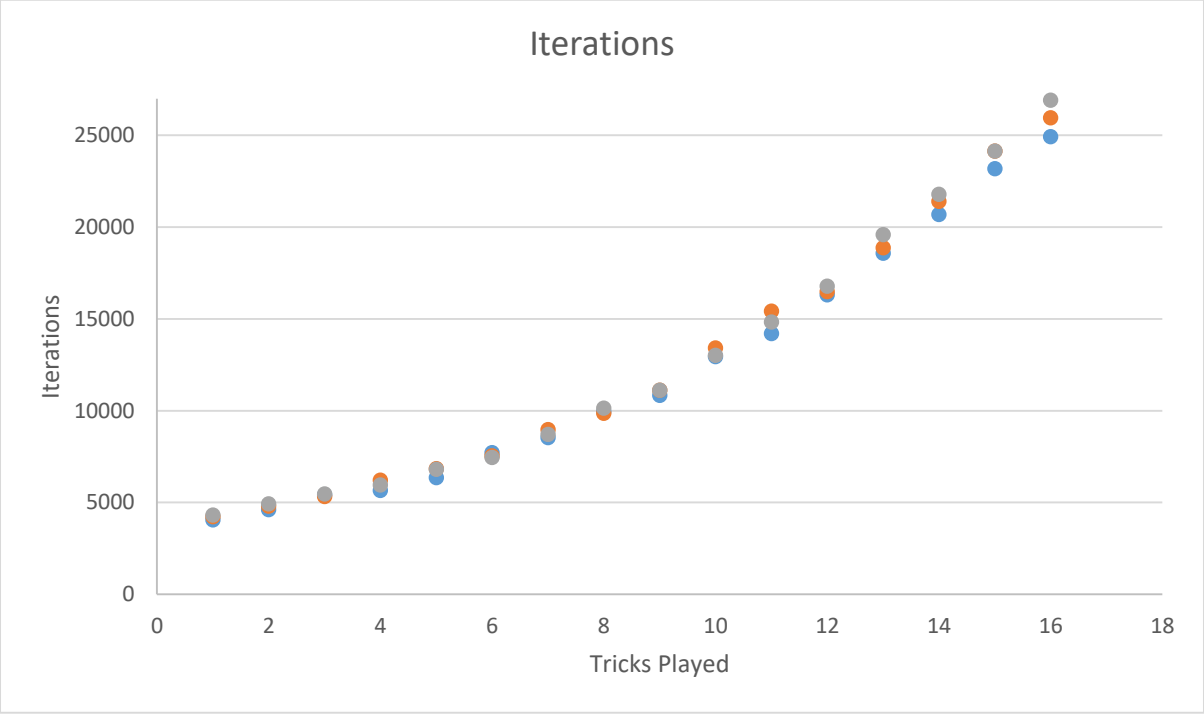
This also demonstrated that any non-random discard policy is exploitable by examining how quickly a suit was "broken" (when a card out of suit was played).

Validation/Conclusion

To evaluate the agent, firstly an optimal agent would have to be found. As there are non-avaliable for the game of Moss Side Whist, it was decided to use a double-blind cheating agent running a normal Monte Carlo Tree Search algorithm, as well as testing against random agents. The tests ran for 30 tournaments of 3 games each (1 game has 3 rounds).

	2 RAND	2 ISMCTS	2 DLS-ESE	2 CHEAT-MCTS
RAND	0.34	-	-	-
ISMCTS	0.43	0.35	-	-
ISMCTS-DLS	0.62	0.50	0.34	0.13
ISMCTS-DLS-ESE	0.755	0.577	0.34	0.25
CHEAT-MCTS	1	0.63	0.45	0.33

Overall my agent performed slightly below expectations, as it was unable to fully beat out 2 random agents and was outperformed to a large degree by the cheating optimal agent. Employing a greedy move handler for opponents (as done by Palma (2014)) showed no performance improvement and in fact showed some signs of regression.



The above graph shows a plot line of the number of iterations of searching done, versus the number of tricks played, with runs under 4000 iterations producing sub-optimal results.

One possible issue may relate to Cowling et. Al (2013)'s observations about bottlenecks when conducting cloning (due to repeated system calls) and move evaluation functions Further steps for improvement would be incorporating greater (rather than the rudimentary) opponent modelling strategies, such as a Bayesian classifier, leverage of parallel processing and additional processing cores of modern CPUs, as proposed by Guillaume et. Al (2008) or using the above as an offline trainer to develop a learning agent.

References

Bethe, P. (2010). The state of automated bridge play.
Available from: http://cs.nyu.edu/~pbethe/bridgeReview200908.pdf

Chaslot, G, Gerritsen, G & Ponsen, M. (2010). Integrating opponent models with Monte-Carlo tree search in poker. Available from: https://dl.acm.org/citation.cfm?id=2908541

Cowling, P, Powley, E, Rollason, J & Whitehouse, D. (2014). Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game. Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2013. Available from:
https://pdfs.semanticscholar.org/7fbb/9070c48397c6ff36cf154c79b37156153ea2.pdf

Frank, I. (1998). *Search and Planning Under Incomplete Information.* Avaliable from:
https://link.springer.com/chapter/10.1007/978-1-4471-1594-6_4

Ginsberg, M. (2011). GIB: Imperfect Information in a Computationally Challenging Game. *Journal of Artificial Intelligence Research*, Volume 14, pp. 303-358, doi:10.1613/jair.820

Guillaume, M, Mark, W & H. Jaap van den, H. (2008). Parallel Monte-Carlo Tree Search. *Computers and Games*, pp 60-71. Avaliable from:
https://dke.maastrichtuniversity.nl/m.winands/documents/multithreadedMCTS2.pdf

Kocsis, L. & Szepesvari, C. (2006). Bandit based Monte-Carlo Planning. Available from:
http://old.sztaki.hu/~szcsaba/papers/ecml06.pdf

Palma, S. (2014). Monte Carlo Tree Search algorithms applied to the card game Scopone.
Available from: http://teaching.csse.uwa.edu.au/units/CITS3001/project/2017/paper1.pdf

Russell, S & Norvig, P. (2016). *Artificial Intelligence: a Modern Approach (Third Edition).* Third edition. Global edition.). Essex, England: Pearson Education Limited.

Smith, S, Nau, D. & Throop, T. (1998). Computer Bridge: A Big Win for AI Planning. *AI Magazine*, Volume 19 (2), pp 93-106. Avaliable from:
http://teaching.csse.uwa.edu.au/units/CITS3001/project/2017/paper2.pdf

Yegnanarayana, B, Khemani, D, & Sarkar, M. (1996). Neural networks for contract bridge bidding. *Sadhana*, Volume 21(3). doi:10.1007/BF02745531