CITS3001 Project Report

Virinchi Rallabhandi
Student Number: 22229206

October 19 2018

1 Literature review and rationale of selected techniques

Despite only being invented eight years ago, I was astonished with Hanabi's popularity for both entertainment and AI research. Through my literature review, I discovered a wide variety of approaches people have taken to building Hanabi playing agents while also considering some options of my own. I found four broad strategies employed - rule based agents, Monte Carlo Tree Search (MCTS), predictive agents and hats agents.[1,2,3,4]

Personally, I find the hats agent most appealing. Hanabi is fundamentally about communication and any method allowing better communication improves performance. "Hats agents" refer to strategies where the players agree beforehand on a method of hinting such that each hint carries more meaning than simply the colours and values of certain cards. They are named after hat guessing games. A famous one involves n people put in a line from tallest to shortest facing forwards. Each is told they have a black, or white, hat on their head, but they are not allowed to look at their hat. Because of the arrangement, they can see all hats in front of them, but none behind. No communication is allowed between the players; they must simply guess their hat's colour. What is the maximum number guaranteed to guess correctly assuming the players agree to an intelligent strategy beforehand? It turns out $n-1$ can succeed. The first player guesses "black" if there are an even number of black hats in front of him/her and "white" if that number is odd. Then, the second player knows the number of black hats in front of them and whether that number should be even or odd once their hat is included. Hence, they guess correctly. The third person knows how many black hats are ahead of him/her, they know the second person guessed correctly and they know whether the number of black hats should be even or odd once their hat is included. Therefore, they guess correctly and so forth. This solution took me six months to find so I can only imagine how long it took Cox et al.[4] to find their solution for Hanabi.

Cox et al. devise two methods - the recommendation strategy and the information strategy. While ordinarily we may think of hints providing information to only one player, the recommendation strategy uses hints to recommend actions to all other players simultaneously. Consider player 4 making a move. Each other player's hand is given a score from zero to seven. A score of $0 \leq i \leq 3$ means the players should play card i. A score of $4 \leq i \leq 7$ means the player should discard card $i-4$.

The hints are also given scores. A score of $0 \leq i \leq 3$ corresponds to a value hint to the

player at position i and a score of $4 \leq i \leq 7$ corresponds to a colour hint to the player at position $i - 4$. Player 4 then gives the hint with score

$$\Sigma \, handScores \; (mod \, 8)$$

Like the hats game, because each of players 0 to 3 knows the hand scores of all players except themselves, they can deduce their hand's score and choose the correct action. The information strategy is more complicated, but in the same spirit. Both strategies work phenomenally with recommendation getting an average score of 23 and the information strategy averaging 24.68.

The hats agent as given by Cox et al. only works for the five player version of Hanabi because it relies on a one-to-one correspondence between hand scores and hint scores. However, they don't make full use of hint options so by changing the way we allocate hint scores, Cox et. al's agent can be extended to a variable number of players. Unfortunately, despite its success, the hats agent is inappropriate for my purposes. My agent will be working with other agents who will not be using the same strategy as me, which would render hats agents useless.

Hence, I must choose between rule-based agents, MCTS, predictive agents or an alternative of my own. All rule based agents essentially have hard-coded heuristics to follow. Osawa presents four types - random, inner state, outer state and self recognition.[2] The random agent is the most boring and the least effective - it randomly selects a move among all legal moves available. It usually blows the fuse and scores zero. An internal state agent keeps track of the possibilities for its own cards based on given hints; it's very similar to the basic agent provided. The only difference between Osawa's internal state agent and the basic agent is a slightly better hinting policy[2]. While basic agent hints randomly, if the internal state agent sees a card which needs to be played, it randomly hints either the value or the colour of that playable card. If no playable card exists then the internal state agent reverts to random hints.

Both random and internal agents do not exploit all the available information. They don't consider the effects of their hints and may give the same hint again. To build a better agent, we must move from these simple reflex agents to a model based agent like the outer state agent. It has a far better strategy which utilises most of the available information and I believe plays much more "humanly." The outer state agent not only keeps a list of possible cards which could be at each position in its hand, it does the same for its teammates' hands (see Figure 1). It plays any surely playable card. If not it discards any surely useless card. If neither is possible, it hints. Unlike the preceding agents, because the outer state agent is aware of its' teammates' knowledge, it only uses hints in ways which improve a teammates' knowledge of a playable card, or a discard-able one if none is available. It's a massive improvement on previous agents.[2]

While the outer state agent uses available hints and information efficiently, it doesn't use fuse tokens efficiently; it never takes a risk. If I know a teammate is using the outer state agent's policy and one of my cards has been hinted, there's a pretty good chance it's playable. This is the basis of self recognising agents,[2] which one could argue are really predictive agents. Each agent assumes its teammates are outer state agents and simulates their behaviour based on different combinations of its own hands. Only the combinations consistent with the actual behaviour observed are kept. Furthermore, if one card is far more likely to be at a position in its hand than any other and it's playable, the agent plays it.

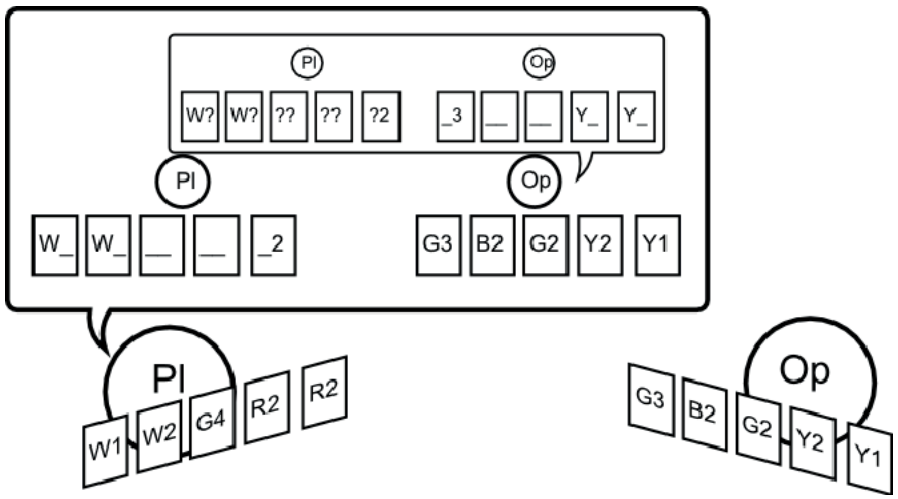

Figure 1: A graphic representing the outer state agent[2] which keeps a list of all possible cards at a position from its opponent's point of view

Most rule based agents in the literature essentially reduce to these agents.[1,2,3] For example, Spieksma et al., have a variation where rather than being completely certain over whether to play or discard, their inner state agent uses probability thresholds, ω_p and ω_d respectively. e.g. if there's a 0.9 chance my card is playable, rather than waiting for another hint to confirm my suspicions, I simply play it. They also test various discard policies - random, most certain to be useless, longest in the agent's hand and least necessary for a firework. Compared to Osawa's inner state agent, Spieksma et al.'s, best agent does significantly better - although Osawa considered two player games and Spieksma et al. considered three player games which are much easier.[1,2] Personally, I believe Spieksma et al.'s analysis was not very complete. Rather than testing some arbitrary values of ω_p and ω_d, I felt a genetic algorithm would have been much better at fine-tuning the probabilities along with the discard policies.

Rule based agents rely on heuristics. Therefore, the quality of the chosen heuristics completely determines their performance, they can never outperform their designer and they don't utilise most of the computational power at their disposal. MCTS is an aheuristic, simulation based strategy using utility based agents which aims to address these issues. I found Di Palma' paper[5] very useful for understanding the process. At each step, we choose a leaf node of the existing game tree and add all its children to the tree. The method used to choose the leaf node is known as the tree policy. From the new nodes we simulate the game, "playout," to a conclusion where each agent plays via a strategy known as the "default policy." The game's result is backpropagated up the tree where each node's utility is adjusted (see Fig. 2). We can use a minimax style algorithm to determine which of the child node's utility is used as the value of the parent; in Hanabi's case, it's just the maximum all the way up. The designer is left to choose the tree and default policies. To stay aheuristic, random agents are conventionally chosen for the default policy. The tree policy is chosen to balance exploration and exploitation.[5] The standard tree policy, given by Kocsis and Szepesvari, goes down from the root and at each

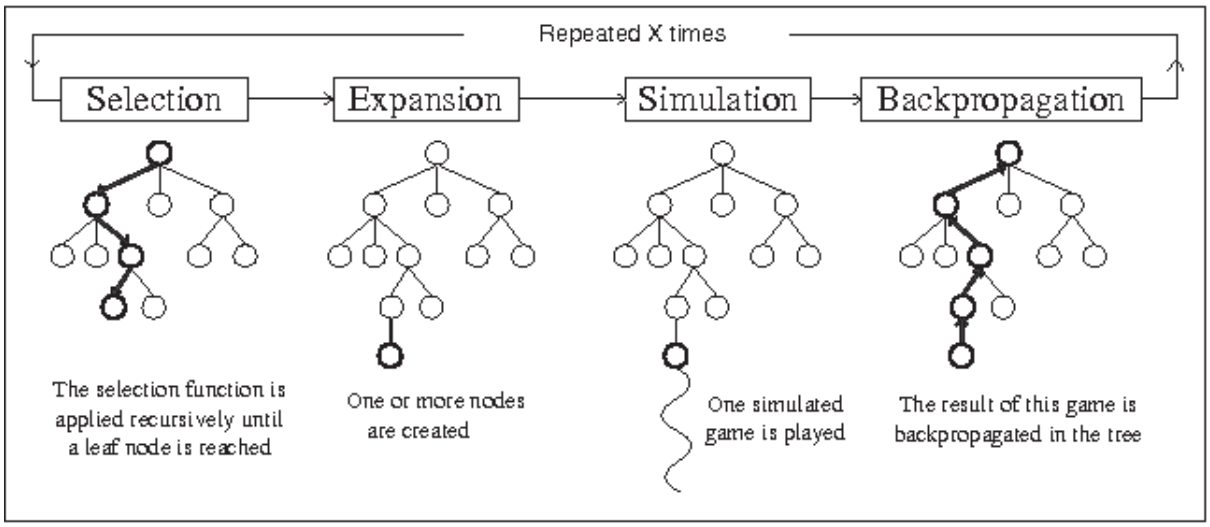

Figure 2: A general model of a Monte Carlo Tree Search[6]

step, selects the the child, n', of node n, which maximises

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

where $N(x)$ is the number of times node x has been visited and $Q(x)$ is the total reward of all playouts passing through x. Hence, the first term represents exploitation while the second represents exploration. MCTS is an anytime algorithm so we use our entire computational budget.

MCTS is harder in Hanabi because of imperfect information. Di Palma suggests determinization - averaging over many random permutations of the deck and possibilities for our hand - as a quick and dirty remedy.[5] A better method is information set monte carlo tree search (ISMCTS) - where each node represents information sets rather than states.[5]

I was quite surprised to find MCTS agents being outperformed by rule based agents.[1,3] Spieksma et al.'s best MCTS agent averaged 14.5 in the three player game where as their best rule based agent averaged 15.4. Similarly, Walton-Rivers et al. also observed their ISMCTS underperforming their rule based agents.

I think the fundamental difficulty of MCTS in Hanabi is branching factor. At each stage we could give any of five different colour hints or five different value hints to any of four other players. Furthermore, we could play or discard any of our five cards. That's a total of 50 legal moves! Hence, MCTS's tree can only be built to a shallow depth. Furthermore, only very few of these 50 moves are any good. Most people use random agents as their default policy[1,3,4] so that their agent is aheuristic - which is really the spirit of Monte Carlo search. Therefore, all MCTS does is pick the best move assuming everyone plays poorly thereafter. That's damage control - not a strategy for ideal play.

I think for games with high branching factor, it may be necessary to abandon the aheuristic part of MCTS. I think a much better strategy, albeit more complicated to code, is using rule based agents such as Osawa's outer state agent as our default policy. Hence we are using both the wisdom of the heuristics together with the computational power available; rule based agents and normal MCTS agents only utilise one or the other, but not both. I went looking to find whether anybody else noticed this problem too. Unfortunately, for my chances of making a ground-breaking improvement, they had. Ponsen et al.[6] used an opponent model - a learned belief system of rational opponent play - to significantly improve MCTS applied to Poker. I think Poker, with its imperfect information, is a valid analogue to Hanabi. Walton-Rivers et al.[1] also managed to modify MCTS to produce a dramatic improvement in performance. Like Osawa's self-recognition agent and Ponsen et al.'s poker agent, they found being able to predict opponent moves better made their Hanabi MCTS far more effective.

In summary, the ideal strategy for Hanabi is the hats agent. While it's a simple reflex agent containing no intelligence of its own, the ingrained strategy is beautiful and by far the most effective known. Other than that, while rule based agents outperform MCTS agents, I think the best AI strategies involve either a predictive agent or an MCTS agent which uses rule based agents in it's default policy. In terms of the rule based agents, the model based agents - like Osawa's outer state agent - outperform all others because they have an understanding of the other agents' information and play according to the needs of their teammates. Therefore, my plan is to implement a model based agent, refine its heuristics, then, if time permits, integrate a model based agent into an MCTS agent. If time does not permit, at least I'm left with a decent agent.

2 Implementation

Creating my model based agent proved to be much more difficult and time consuming than I imagined. The main data structure required was a triply nested list. list[i][j][k] contains the k^{th} card which could be at the j^{th} position of the i^{th} player's hand in the i^{th} player's point of view. Here, the number of values for i is fixed - it is the number of players - where as the number of values for j and k is not because cards could become null at the end of the game and we learn more about which cards we have as games progress. Therefore, I really want an array of doubly nested ArrayLists. However, one can't create an array of parameterised types. A simple solution was to make an inner class - AgentInformation - which does nothing more than extend the ArrayList<ArrayList<Card>> class. It also made my code easier to read because it separated the player and possible information sets into different looking data structures. My final agent uses the following policy.

1. If I have deduced the identity of a playable card, I play it.
2. Else, if I have deduced the identity of a useless card, I discard it.
3. Else, if one of my cards has been partially hinted - i.e. I only know the colour or the value - then I play it provided a mistake won't blow the fuse.
4. Else, if there are available hint tokens, I look through all the other players' cards and think about hinting the first playable card. If the player holding that card doesn't know its value, I hint the value.

```
   Else, if they do not know its colour, I hint its colour. If they know
   both, I go looking for a different playable card.
5. If I was not able to find a playable card, I repeat step 4 but looking
   for useless cards which should be discarded.
6. If I was not able to find a player who is unaware they are holding a
   useless card, I give a random hint if there are more than 4 hint
   tokens available. Else, I discard the card in my hand for which
   I have the least information - i.e. the position where my possible
   cards set is largest.
```

All operations are $O(1)$ - albeit with large hidden constant - because the number of players and the number of cards is bounded. The policy was developed both by experimentation (see validation) and by considering how I would play the game myself. At the start of each action, the agent processes information gained from the preceding set of moves from each player's point of view. Then it removes all impossible cards - either because of hints, discards, plays or visibility in someone else's hand - from each information set in the triply nested list.

I hoped to incorporate this model based agent as the default policy of a MCTS. Unfortunately, it took me an age to debug the model based agent - I was amazed such a simple looking agent could require 700 plus lines of Java - and I was left with very little time to develop MCTS. I've included partial progress in the zip file.

3 Validation

The only real way to test agents' performance is to run many games, with randomised initialisation, and average over the final scores in each one. For game-playing agents, scores' standard deviation only corresponds to the "riskiness" of a strategy. If a riskier strategy pays off more often than not, we should play it anyway i.e. only mean is relevant here. I only collected standard deviations for purposes of p-value testing of statistical significance via the central limit theorem. As for the actual games played, I see only two parameters - the number of players per game and an agent's peers in the game; some agents may only work well when playing with other similarly intelligent agents or agents using exactly the same strategy. To summarise results efficiently, I use the tuple (mean, standard deviation, number of players). All tests were run with 100 thousand trials unless otherwise stated. My final agent scored (15.961, 1.988, 5), (17.024, 1.996, 4), (17.047, 1.881, 3) and (15.314, 2.821, 2) while playing with other identical agents. When my agent played with only basic agents, the scores were (8.765, 4.411, 5), (9.754, 5.283, 4), (10.540, 6.052, 3) and (11.116, 6.405, 2). The high standard deviation occurs because the basic agent keeps blowing the fuse.

I think it's worth mentioning some variations to my model based agent's policy (see implementation) which I considered during development. Initially I didn't have step 3; I was very surprised to find it making a major difference to my agent. Without step 3, when playing with identical agents, the scores were (13.913, 1.089, 5), (15.505, 1.213, 4), (16.206, 1.418, 3) and (14.730, 3.453, 2) which is a statistically significant dip - p-value $\ll 0.01$ - in all cases. In hindsight, it makes some sense. Without step 3, the agent doesn't make full use of the fuse tokens. I was still reluctant to take these risks. To me it seemed playing riskily would only be

profitable in the early or late game. I chose to control at which points in the game step 3 was active and found it was best to never turn it off at all (see ModelAgent3 in zip file).

Step 3 relies on agents hinting mostly only playable cards; I think that's a fair assumption to make of moderately intelligent agents. In fact, even while playing with only basic agents, who hint randomly, without step 3, the scores were (8.597, 3.498, 5), (9.702, 4.069, 4), (11.098, 4.152, 3) and (12.635, 3.763, 2) - a statistically significant improvement in the two player version, a statistically significant regression in the three and five player versions and a statistically insignificant regression in the four player version compared to my final agent.

I also, considered making step 1 "play or discard the first playable or useless card respectively identified to be in my hand." That scores (15.886, 2.002, 5), (16.911, 2.015, 4), (16.935, 1.912, 3) and (15.212, 2.818, 2) - small, but statistically significant, regressions in all cases. Similarly, I tried putting steps 4 and 5 together - i.e. look for a playable or discardable card. This fails sensationally with scores of (11.434, 1.521, 5), (12.540, 1.667, 4), (13.304, 1.755, 3) and (13.735, 2.567, 2). Buoyed by these results I considered switching steps 3 and 2 which scored (15.902, 1.996, 5), (16.871, 1.977, 4), (16.858, 1.813, 3) and (15.177, 2.817, 2) - slightly less in all versions at a statistically significant level.

Finally, I tried three discard policies; seeing as agents will be forced to blindly discard often, it's an important choice. I tried discarding randomly, discarding the card least likely to be useful and discarding the card for which I have least information - the policy I finally adopted. The random discard was not too bad, with scores of (15.437, 2.167, 5), (16.224, 2.206, 4), (16.186, 2.030, 3) and (10.197, 4.614, 2), but I was surprised to find the least likely useful heuristic wasn't the best. It scored (15.196, 2.337, 5), (15.783, 2.403, 4), (15.443, 2.201, 3) and (11.306, 3.298, 2).

Note that all results were as of 17^{th} October. There were some minor modifications to the GitHub game files after this date which may affect results slightly.

4 References

1. Walton-Rivers J, Williams PR, Bartle R, Perez-Liebana D, Lucas SM. Evaluating and modelling Hanabi-playing agents. InEvolutionary Computation (CEC), 2017 IEEE Congress on 2017 Jun 5 (pp. 1382-1389). IEEE.

2. Osawa H. Solving Hanabi: Estimating Hands by Opponent's Actions in Cooperative Game with Incomplete Information. InAAAI workshop: Computer Poker and Imperfect Information 2015 Apr 1 (pp. 37-43).

3. Spieksma FM. Aspects of the Cooperative Card Game Hanabi. InBNAIC 2016: Artificial Intelligence: 28th Benelux Conference on Artificial Intelligence, Amsterdam, The Netherlands, November 10-11, 2016, Revised Selected Papers 2017 Sep 14 (Vol. 765, p. 93). Springer.

4. Cox C, De Silva J, Deorsey P, Kenter FH, Retter T, Tobin J. How to make the perfect fireworks display: Two strategies for hanabi. Mathematics Magazine. 2015 Dec 1;88(5):323-36.

5. Di PALMA ST. Monte Carlo tree search algorithms applied to the card game Scopone.

6. Ponsen MJ, Gerritsen G, Chaslot G. Integrating Opponent Models with Monte-Carlo Tree Search in Poker. Interactive Decision Theory and Game Theory. 2010 Feb;82.