CITS3001 Algorithms, Agents and Artificial Intelligence Project

By Ryan Hodgson (21969062) and Jonathan Kok (20744321)

1. Introduction

Hanabi is a cooperative card game with incomplete information. Each player is able to view every other player's hand, but not their own. The goal of the game is to complete five separate stacks representing different coloured fireworks. A player has the ability to play/discard a card or give a hint to another player.

The inability to access all information within a game brings its unique challenges. This paper explores the possible strategies that help combat this limitation and explores the selected strategies for our own agents playing the game.

2. Literature Review Of Suitable Techniques

2.1 Incomplete Information AndIts Effect On The Game

When participating in a game of Hanabi an agent is able to access every other player's hands, but not its own. The agent has *incomplete information* when assessing what action will maximise total utility for the game.

Humans are able to alter their perception of their (inaccessible) hand and note which cards are seemingly worthless or of value based on the behaviour of others (Osawa, 2015). An agent is limited in this respect.

An agent needs to acquire information from each action performed by every other agent in the game.

When a player gives a hint, it is important to recognise that this encounters a cost of one hint token.The information gained should be considered carefully. A hint gives two pieces of information. Those cards that *are*, and those cards that are *not*.

A discard can be interpreted that the player does not have enough information to play confidently or enough information has been given to discard a card.

Various articles offer different approaches to handle this restriction. In her paper on AI strategies for Hanabi, Hirotaka Osawa offers an outline for a “self-recognition” strategy (Osawa, 2015). This is built on top of a rule-based agent as described below.

2.2 Rule-Based Approach

When playing Hanabi, intuitively a number of rules become apparent when attempting to maximise the end score of the game.

When all fireworks are empty, any card with the value of “one” can be played

When all fireworks stacks are above a certain value, any cards below/equal to this value can be discarded

There is only a single five for each colour so they must not be discarded when in a player’s hand

2s, 3s and 4s all have duplicates – however if any one of these is discarded the remaining card must be treated appropriately.

These rules can be hard-code quite easily into a simple reflex agent. However, issues arise when introducing the concept of hints. Which hint will maximise the final payoff? How is an agent meant to interpret a hint? An agent is limited by the number of hint tokens available too.

The approach considered and implemented was that of a conservative/cautious agent which would follow a hierarchy of actions.

Play a card that is guaranteed to work

Give a hint to another players card which should maximise payout

Discard a card

Hints were prioritised by the amount of information that can be obtained from them. It was important that discarding was limited as it was a risk and could limit the maximum score obtained.

For every state, a utility was assigned to each card based on the information known.Therefore, the current agent was able to identify (based on the above rules) which of their cards and other players cards were important, and which were not.

Furthermore, hints to other players were checked against a memory bank so as to make sure agents weren't wasting hint tokens on giving the same hints.

2.3 Self-Recognition Strategy

Since Hanabi is a cooperative game it is in each player's own interest to play rationality. That is, to play so that each action attempts to maximise utility. To predict other agent's actions, it is not unreasonable to simulate our own rational agent and record what actions it took. Any agents which are uncooperative will harm both everyone else's utility and its own, an undesirable outcome.

The Self-Recognition Strategy assumes each agent has the same strategy. It also has a similar strategy to the rule-based agent with an additional step explored below.

As Hirotaka Osawa's study outlines, initially every possible hand our agent could have is generated. We then simulate the action the previous player would have played, given each of these hypothetical hands. If the hypothetical action is the same as the actual action the hand is kept in memory, otherwise it is discarded. This continues until all hypothetical hands have been exhausted.The two cards with the greatest chance of occurring within the agent's hand is stored in the variables x and y. If $\frac{x}{y} > a$ then we estimate that this card is within the player's hand. The study found that a threshold value (a) of 2.5 is optimal (Osawa, 2015).

An agent with this strategy infers the state of their hand by using the assumed behaviour of others to interpret new information. This agent appears to act humanly.

2.4 Monte Carlo Tree Search

The Monte Carlo Tree Search (MCTS)has four major components.

The *selection* action involves traversing the tree from the root node to a leaf node (a node where no simulation has occurred). The child node which maximises the value returned by the UCB1 formula is chosen. This formula is used to balance between the*exploration*of nodes with minimal visits and *exploitation* of nodes with a high average value.

$$UCB1(S_i) = \overline{V_i} + c\sqrt{\frac{lnN}{n_i}}$$

$$\overline{V_i} = average\ value\ of\ the\ current\ state$$

$$N = number\ of\ visits\ of\ the\ parent\ node$$

$$n = number\ of\ visits\ of\ the\ current\ node$$

$$c = exploration\ parameter$$

When a leaf node is reached the *expansion stage* adds all valid actions as children. The *simulation stage* involves randomly playing out one round of the game. The result of this is used to update information along all nodes from the respective node to the root. This process is known as *backpropagation*.

The MCTS seems attractive and viable for Hanabi for several reasons.

An evaluation function takes an input of a valid move and produces an estimate of the cost the move would incur on the game. Hints complicate the process of creating an efficient evaluation function. The definition of a good or bad hint varies between players since a hint requires some level of inference from the agent receiving the hint. The MCTS has no need for an evaluation function and needs only to be able to interpret the outcome of a game (Levine, 2017).

A greater runtime will expand the size of the tree the MCTS is traversing and improve thereliability of the average value of each node. However, the MCTS can be stopped at any time and it will still return the best move it has found so far.

The Monte Carlo tree grows asymmetrically, exploring those nodes which appear promising. This allows the algorithm to tackle problems with a high branching factor.

It would be ideal to run the MCTS with perfect information. Since this is not the case, what strategies can we employ to maximise the accuracy of the MCTS?

A paper by Markus Egar *et al.* outlines what is called the "Mental State Representation" for keeping track of possible identities for each of their cards.Every time a hint is given the information is translated to this table (Eger et al., 2017).

Table 1.1 demonstrates what such a table would be initialised to. Every time the agent recognises that it does *not* have certain card it will reduce the corresponding value.

Rank	Red	Green	Blue	White	Yellow
1	3	3	3	3	3
2	2	2	2	2	2
3	2	2	2	2	2
4	2	2	2	2	2
5	1	1	1	1	1

Table 1.1: Initialized Hanabi mental state representation table.

From this we can determine the probability that a card is what we think it is (e.g. in the above table $P(Red1) = \frac{3}{52}$).This does require multiple rounds to be have been played however, so enough information can be accumulated.

3 Rationale Of Selected Technique

Ultimately, two Hanabi AI agents were finalized and implemented. The primary agent implements a simple version of the outer-state strategy. Like the internal-state strategy, the agent memorizes the hints it receives and will play a card it knows is playable. It will also prioritize cards it knows are safe to discard when it chooses to discard. When it comes to giving hints, the agent will recognize cards that are currently playable in another player's hand and give hints towards that card. Additionally, the agent will remember all hints that have been given in the game and will not give duplicate hints. It is thus able to identify priority cards in another player's hand and give complimentary hints if that player has partial knowledge on the card. However, the agent will *only* play cards it knows the colour/value of and knows it is playable. Since it will never make a risky or uncertain play, we have called the agent "ConservativeAgent".

The other completed agent "AdvancedAgent" is based on the ConservativeAgent but with limited capacity to make risky moves. If the agent has a card with partial knowledge, it will look at all viewable cards (discard pile, fireworks pile, other player hands and known cards in own hand) and attempt to guess the unknown attribute of the card. If it fails to find a single answer, but finds that there is a 50% chance that the card is playable, it will play the card anyway if there are fuse tokens to spare.

Self-Recognition and Monte-Carlo Tree based agents were prototyped and explored, but ultimately dropped. The Monte-Carlo Tree method requires simulating the actions other players will take, but without the knowledge of the player’s hand it cannot accurately simulate another player, as it would be simulating a round where one player’s hand is missing. The value of such an uncertain and incomplete simulation is thus questionable.

The Self-recognition strategy involves analysing the previous action (when given a hint) and from the superset of all possible hands, narrow it down to the set of all possible hands that fit the hint that was given. This was impractical primarily due to the excessive computation required to generate even a subset of all known hands.

Additionally, the flaw of all probability-based agents is the limitation of fuse tokens: the worst possible outcome in a game is to run out of fuse tokens and receive 0 score. As there are only 3 fuse tokens with the game lost at 0, that leaves only 2 chances for mistakes shared across all players. Once any two mistakes have been made, all agents *must* then default to the conservative outer-state strategy to avoid total failure. Given this limitation it is clear that there are diminishing returns to relying too heavily on probability-based strategies as there is a high chance that such strategies would need to be discarded early in the game, or if the acceptable risk probability is low then would result in similar performance to the conservative agent.

4 Implementation Description

The ConservativeAgent prioritises actions in the following order: playing a playable card, giving a useful hint or discarding a card.

The ConservativeAgent will only play cards it knows both the value and colour of, and only if they are currently playable. To that end it uses two complimentary functions: stacksInfo() and currentHandUtility(). The former function creates a HashMap that essentially records the current value for each firework stack, how many empty stacks there are and the "minimum" value that is currently placed on any firework stack. This information is passed to currentHandUtility which thus assigns priority to each card in the player's hand based on current hints. Playable cards have the highest priority, but also any unplayable cards (cards with value below the minimum threshold) are de-prioritized to be discarded first.

If no card in the agent's hand is confirmed playable, the agent looks at each other player's hands. The function otherHandUtility() applies a similar priority-assignment to each other player's hand with playable cards having the highest priority, and secondarily considering hints that will reveal the most information about a hand (hint applied to the most cards). If there are hint tokens available, the agent will then prioritise giving hints to other players towards these cards.

The agent also keeps a "memory" of every hint that has been given in the game. This is saved as a HashSet of String values. Every hint in the game is converted to a string of the format:

```
{
        [hint receiver index]
        [0 -> hint about colour, 1-> hint about value],
        [colour or value of the card],
        [boolean array of hinted cards]
}
```

These hints are stored in the HashSet and before any hint is given, the memory is checked to ensure it does not give duplicate hints.

If no valuable hints can be given, the player discards a card, prioritising those "safe" to discard first.

The AdvancedAgent keeps a memory of the hints every other has at the time in an array, similar to the player's hints. This was used to create another function that can recognize another player needing complimentary hints to recognize their cards. The main feature of this agent however, is the "thinkEngine()" function. This function is able to count all cards that can be seen (in each player's hands, in the agent's hands if hints allow, discard pile and fireworks stack) in an attempt to discern the complimentary information of a card they only have partial knowledge of (as there is only 2 of any card/value combination, with the exception of 1 and 5 cards). If there is a 50% chance it knows what the card is and one of those possibilities is playable, it will play the card if there are spare fuse tokens.

5 Validation

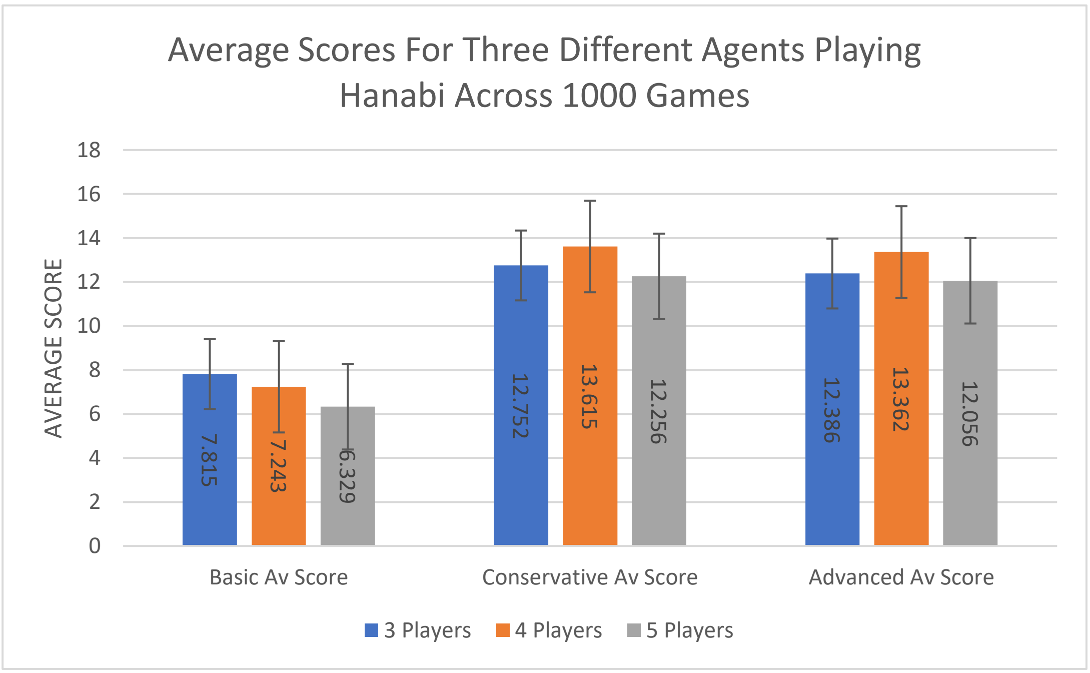

Figure 1.2: Average scores between three Hanabi agents over 1000 games. *Each data bar lists the numerical average and displays the standard deviation for the score.*

Figure 1.2 demonstrates the average performance of three agents across 1000 games of Hanabi. Both the Conservative and Advanced agent performed at a higher standard overall than the Basic agent. This can be contributed to the fact that the Basic agent has no concept of its environment and relies on chance (randomness) for its next move.

No Of Players	3	4	5
Basic Av Score	7.815	7.243	6.329
Conservative Av Score	12.752	13.615	12.256
Advanced Av Score	12.386	13.362	12.056

***Table 1.3: Average scores for 3 Hanabi agents over 1000 games**. Each agent was tested in an environment of 3/4/5 players of the same AI type as itself.*

It is interesting to note that the Advanced agent did not perform significantly different from the conservative agent. As previously discussed, probability-based strategies are limited by the combined 2 mistakes per-game for all players limit. Adjusting the AdvancedAgent to act on riskier moves tends to lead to the 2 mistake limit being reached early, while lowering the risk tolerance results in few risky moves attempted. The end-result is that even a risk-taking agent will default to the "conservative" strategy for the majority of the game, resulting in scores similar to conservative-rule agents.

Both the Advanced and Conservative agent perform optimally for a four-player game. A five-player game may involve more inference from each agent as there are more players sharing the same number of hint tokens. This involves a greater degree of guessing and hence risk. A three-player game on the other hand may limit the amount of information that is transferred to each player in a round. The larger the number of agents that play, the greater the chance you are going to receive a hint and hence, information. However, this is of course still limited by the number of hint tokens you have.

6 References

EGER, M., MARTENS, C. & CÓRDOBA, M. A. An intentional AI for hanabi. Computational Intelligence and Games (CIG), 2017 IEEE Conference on, 2017. IEEE, 68-75.

LEVINE, J. 2017. *Monte Carlo Tree Search* [Online]. YouTube. Available: https://www.youtube.com/watch?v=UXW2yZndl7U [Accessed 21/10/2018].

OSAWA, H. Solving Hanabi: Estimating Hands by Opponent's Actions in Cooperative Game with Incomplete Information. AAAI workshop: Computer Poker and Imperfect Information, 2015. 37-43.