

Evaluating and Modelling *Hanabi*-Playing Agents

Joseph Walton-Rivers* Piers R. Williams† Richard Bartle‡ Diego Perez-Liebana§ Simon M. Lucas¶

School of Computer Science and Electronic Engineering,

University of Essex, Colchester

CO4 3SQ, UK

Email: {jwalto*, pwillic†, rabartle‡, dperez§, sml¶}@essex.ac.uk

Abstract—Agent modelling involves considering how other agents will behave, in order to influence your own actions. In this paper, we explore the use of agent modelling in the hidden-information, collaborative card game *Hanabi*. We implement a number of rule-based agents, both from the literature and of our own devising, in addition to an Information Set-Monte Carlo Tree Search (IS-MCTS) agent. We observe poor results from IS-MCTS, so construct a new, predictor version that uses a model of the agents with which it is paired. We observe a significant improvement in game-playing strength from this agent in comparison to IS-MCTS, resulting from its consideration of what the other agents in a game would do. In addition, we create a flawed rule-based agent to highlight the predictor’s capabilities with such an agent.

I. INTRODUCTION

Hanabi is a co-operative, partially-observable [1] board game which in 2013 won the prestigious *Spiel des Jahres* award for best board game of the year. For the reasons outlined below, it has featured in a number of recent academic publications. This paper explores whether the use of agent modelling can lead to an improvement in strength for agents playing the game.

Hanabi has a number of interesting features that make it a good choice for research in the field of agent modelling. Firstly, the domain is a co-operative one, in that the agents must work together to achieve a shared goal. This disfavors agents that behave greedily: for example, helping another player score a point is better than playing a risky card that might end the game. Secondly, its rules build in well-defined communication actions. These use a resource that regulates communication and must be managed by the agents. Finally, the game has hidden information, with no one player able to see the entire game state. This is a source of complexity for agents, because imperfect information needs to be reasoned about intelligently. Note that *Hanabi* has been proven to be NP-Complete even when players have perfect information [2].

A number of rule-based approaches for designing agents that can play *Hanabi* have been presented in the literature, however there have been few attempts to employ more general strategies. In this paper, we go some way towards redressing this imbalance. Furthermore, because the use of information about other players’ strategies can help to inform human players in co-operative games, we also explore whether or not such information could help guide our general agents’ decisions.

Section I-A describes the rules of the game of *Hanabi*.

Section II describes the agents that were implemented to play *Hanabi* under these rules.

Section III describes how the agents were tested and evaluated.

Section IV presents the results of the tests.

Section V discusses and explains our findings in the results.

Section VII discusses potential future AI-related work involving *Hanabi*.

A. *Hanabi*

Hanabi is a co-operative game in which a team of two to five players attempts to complete five stacks of sequentially-numbered cards (one for each of the game’s five suits).

The game is played with a deck of 50 cards, each possessing a suit and a rank. The suits are coloured white, yellow, green, blue and red. Within each suit, there are three cards of rank 1, two cards each of ranks 2, 3 and 4, and one card of rank 5. The game additionally features two types of token: an *information token* and a *life token*. The players collectively start the game with 3 life tokens and 8 information tokens.

Every player begins with a randomly-dealt hand of five cards. Cards are held facing away, such that players can’t see the suit or rank of their own cards but can see the suit and rank of the cards held by the other players. The cards not dealt out at the start are placed face down as a *draw deck*, which will be accessed during play.

Play proceeds with each player taking it in turn to perform an action of their choice. There are three different types of action available:

- Tell** Select a player and point to all their cards of a given number or suit. This costs one information token.
- Play** Choose a card from the player’s own hand and play it.
- Discard** Choose a card from the player’s own hand and add it to the discard pile

In a single Tell action, a set of cards can only be identified either by their suit or by their rank — not by both. Furthermore, cards must be present in the hand to be identified: it is not permitted to state that another player has no cards of a given suit or rank.

Playing a card means adding it to the stack with matching suit. It is not required that the player know which stack it belongs to - for example, at the beginning of the game it is acceptable to blindly play a 1 that has been indicated to you. Each card in the stack must be of the correct suit and have a

rank one greater than the card below (except for 1 cards, which are used to start a stack). If a card is played out of sequence, the group loses one life token. Completing a stack of cards associated with a given suit grants an additional information token (if the team does not already have the maximum number, eight).

Discarding is only permitted if there is at least one information token to be gained. This means that either a Tell action or a Discard action is always possible.

After either discarding or playing a card, the player draws a replacement card from the draw deck. Discarded cards are visible to all players. Discarding a card increments the number of information tokens up to the maximum. Once all cards in the draw deck have been drawn, all players get one more turn and then the game is considered to be over. The game also ends if the team uses up all of the life tokens.

Scoring is achieved by summing the top card of each stack that has been correctly played. The maximum possible score for the standard game is therefore 25, obtained by completing the stacks for all five suits. Remaining life or information tokens are not counted towards score in the standard game.

B. Multi-agent domains

Multi-agent domains can be categorised as either *centralised* or *distributed*. A centralised system features a single controller controlling multiple agents; a distributed system has each agent in the world controlled by a separate controller. In this paper, we consider only the distributed approach.

Existing work in this space includes: attempting to reason about what the other agent knows using answer set programming [3]; iterating on a plan that is communicated between agents [4]; and attempting to use plan recognition to allow one agent to assist another in a planning task [5].

Another possibility involves co-operative, multi-agent learning. Within this area, there have been attempts to learn models of teammates in order to make more informed decisions about which action to take. For a review of the literature, see Panait & Luke [6].

The use of embedding agent models into Monte Carlo Tree Search (MCTS) has previously been looked at by Barrett *et al* in the pursuit domain [7]. They made the assumption that all agents except for their modelling agent would be using the same, fixed strategy, and embedded perfect knowledge of this strategies into their agent. One of their findings was that the system did not perform well with models that didn't represent the behaviour of the agent.

The use of Theory of Mind (ToM) (reasoning about what the other agents know and will do in a given situation) has proven useful in competitive games such as Rock Paper Scissors [8]. In these games, higher-order ToM agents were able to out-perform lower-order ToM players.

C. Co-ordination in Hanabi

In *Hanabi* all agents have access to different information; because of this, a centralised approach to multi-agent planning would not make sense in this domain as private information must not be shared between agents.

The fact that the Tell action has an associated cost (an information token) means that information about a player's hand needs to be communicated efficiently. Also, because *Hanabi* players are limited to a set of well-defined communication actions, communication between them is very limited. This makes using communication between agents to co-ordinate their actions a challenging prospect — which is one reason why *Hanabi* is increasingly becoming the object of research.

The understanding of other players' strategies forms a core component of a great number of games and has been studied widely [8]. Existing *Hanabi* research assumes that all agents are playing the same pre-agreed strategy. The ability to reason about the actions that a player would take and their reasons for taking these actions can be used as part of the reasoning process of an agent.

Our approach is to assume that we have access to a model which, given a state, will be able to return a possible action that an agent would perform in that state; if the agent may make multiple moves, then a single action from the set of possible actions will be returned. Given this model, we are able to incorporate the behaviour of the other agent into our model without understanding of that agent's reasoning process.

A point to note is that Tell actions can convey more information than just the obvious: because *all* cards of a given suit or rank must be identified, cards which are not identified therefore must not satisfy the criterion. This *negative information* can be used to inform the possible values for a given card. Negative information can add up over a few turns, providing enough information to determine what a card is — or at least that it is playable. In the end game, such knowledge becomes very powerful.

Human players of *Hanabi* often make additional use of Tell actions. In particular, they can restrict their Tell actions by convention only to identify certain cards as playable. For example, suppose that Player 2 had the hand $\{(R, 1), (B, 1), \dots\}$ and the current stacks on the table were $\{(R, 1), (B, 0), (G, 0), (W, 0), (Y, 0)\}$. Player 1 may elect to tell Player 2 about the suit rather than the number, to avoid identifying the non-playable red card. Player 2 could then infer that the card being identified was indeed a playable card, as they would know that Player 1 would not have identified a non-playable card. As they were told the suit rather than the number, they could further infer that they have a non-playable 1 in their hand (although they would not know the location of this card). The use of information in this way requires an understanding of how the player will use the provided information as part of their policy.

D. Monte Carlo Tree Search

MCTS [9] is a widely-used tree-search algorithm that can operate without domain-specific knowledge. This gives MCTS the *anytime* property: the algorithm can be stopped at any time and can provide an answer for the next move. Given more time, it will typically produce a more accurate answer.

MCTS proceeds using multiple iterations of the four main stages shown in Figure 1. The iterations typically continue until a predetermined end condition is met, such as running

out of time. In the selection stage, the current tree is traversed using the tree policy to select the best child of each node. In the expansion stage, a new node is added to the tree. In the main, simulation phase, a simulation (*rollout*) of future moves is undertaken from the state represented by the new node until an end condition is met. Moves are selected according to the default policy (which is often to select at random from all possible moves). In the backpropagation phase, nodes in the tree that were selected are updated with the result of the rollout.

E. Monte Carlo Tree Search and Theory of Mind

Zero-order theory of mind [8] agents are capable of using an agent’s history in order to inform future actions. A first-order theory of mind agent is capable of using a model of a zero-order theory of mind agent to inform its own future decisions. Our selection of MCTS for use in this domain came from a particular desire to find an algorithm that could be easily modified to operate with predictions of what other agents would do. This makes it a zero-order agent.

This approach has been tested before in the Tiny Co-op domain [11] by Walton-Rivers, who found that prediction worked best with a deterministic agent that did as it was instructed [12]. The Tiny Co-op domain is a simple, grid-based world containing a number of agents, goals, doors and buttons. Each agent must visit each goal individually for successful completion. Doors separate different areas in which the agents can move, and each door will only open if an agent is standing on its associated button. This forces the agents to co-operate to succeed overall.

While MCTS was a good performer in Tiny Co-op when paired with itself (and even with random agents), it struggled when trying to co-operate with a particular agent that was designed to follow direction indications. Essentially, this *follower* agent moved to where it was instructed to move, but MCTS didn’t pick up on this. The root cause was that it didn’t model such behaviour in its search tree, leading to inaccurate states in the majority of the search space. The author added agent modelling to MCTS and found that the performance of MCTS when paired with the follower agent improved significantly. In this paper we used this approach to create a *Hanabi*-playing agent to assess the effectiveness of agent modelling in this domain.

F. Previous research

1) *Imperfect Information AI*: Games with imperfect information are a complex challenge for AI. Poker is often chosen as an application, because it is a game that many people are familiar with on some level. Poker contains an unusual dynamic for games, as a strong player doesn’t so much play the game as play the opponents. Winning requires a player to understand their opponents and to adopt a strategy that will counter their strengths while exploiting their weaknesses. Rule-based agents feature strongly in this, as do simulation-based agents such as MCTS. Poker has been extensively studied — see the review conducted by Rubin & Watson [13];

one of their notable finds was that a simulation-based approach is inferior to the formula-based approach, despite expectations.

Whitehouse *et al* [14] looked into using MCTS for the card game Dou Di Zhu, which (like *Hanabi*) also features imperfect information. Here, they apply determinisation and IS-MCTS to the problem and conclude while the IS-MCTS is superior in some cases, no overall difference was observed.

2) *Hanabi AI*: There has been a small amount of research into using artificial intelligence techniques to play *Hanabi*. Osawa [15] devised a number of rule-based agents for the 2-player version of the game, the mechanisms for which are described in Sections II-A2 and II-A3. Osawa found that the incorporation of consideration of the other agent’s strategy and why they did what they did allows an agent to perform better than do the other non-cheating agents.

Cox [16] derives strategies for the game of *Hanabi* using the *hat guessing game* as inspiration. The agents all use an agreed encoding strategy to indicate what any particular Tell action specifically means, enabling them to co-operate so as to work around the limited view of their own hands. The encoding strategy does require the 5 player version of the game, however, as it won’t work unless the hand size matches the number of other players in the game. We considered using this agent in the tests, as its unique strategy could have been the perfect test for agent modelling. However, there is an issue with the encoding strategy: every agent must know what every other agent has in their hands. This is cannot be used in agent modelling. If the Predictor IS-MCTS is agent 1, then it has access to the hands of agents 2, 3, 4 and 5 — which in *Hanabi* it does indeed have. Unfortunately, its internal copy of agent 2 needs access to the hands of agents 1, 3, 4 and 5. Agent 1 cannot give this information without breaking the rules of *Hanabi*. For this reason, we did not run tests with this agent.

Van den Bergh *et al* [17] analyse *Hanabi* and define a number of rules for the game. The amount of time it would take to test every possible combination of these rules was too large, however, so they used an iterative approach to explore the search space intelligently. They note that some rules are far more effective than others, as well as observing that a risk-taking rule does have some value. They found that the use of a Discard action when there is a possible hint is not optimal. In a follow-up paper [18], the authors present their best rule-based agent along with one using a Monte Carlo search.

II. AI

A number of the controllers used in this experiment were implemented as production rule agents. Many of these share individual rules, so each rule will be described here independently. All rules have additional pre-conditions that ensure they can only fire if it is legal to do so within the game rules (for example, a Discard action would necessitate a check that an information token was available). To avoid verbosity, we assume that the rules of *Hanabi* will be properly followed (so, for example, if the rule says to inform a player about a card, then the player will also be informed about other cards that satisfy the Tell’s stated criterion).

- **PlaySafeCard**: Plays a card only if it is guaranteed that it is playable

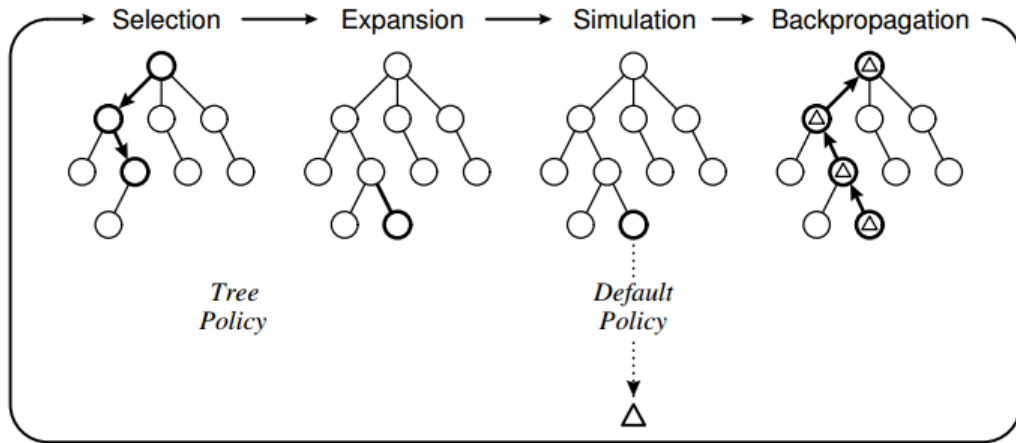


Fig. 1: The four steps of Monte Carlo Tree Search [10]

- **OsawaDiscard:** Discards a card if it cannot be played at the end of the turn. This will discard cards that we know enough about to disqualify them from being playable. For example, a card with an unknown suit but a rank of 1 will not be playable if all the stacks have been started. This rule also considers cards that can not be played because their pre-requisite cards have already been discarded.
- **TellPlayableCard:** Tells the next player a random fact about any playable card in their hand.
- **TellRandomly:** Tells the next player a random fact about any card in their hand.
- **DiscardRandomly:** Randomly discards a card from the hand.
- **TellPlayableCardOuter:** Tells the next player an unknown (to that player) fact about any playable card in their hand.
- **TellUnknown:** Tells the next player an unknown fact about any card in their hand.
- **PlayIfCertain:** Plays a card if we are certain about which card it is and that it is playable.
- **DiscardOldestFirst:** Discards the card that has been held in the hand the longest amount of time.
- **IfRule(λ) Then (Rule) Else (Rule):** Takes a Boolean λ expression and either one or two rules. The first rule will be used if the λ evaluates to true. If it is false, and a second rule was provided, then that will be used instead.
- **PlayProbablySafeCard($Threshold \in [0, 1]$):** Plays the card that is the most likely to be playable if it is at least as probable as *Threshold*.
- **DiscardProbablyUselessCard($Threshold \in [0, 1]$):** Discards the card that is most likely to be useless if it is at least as probable as *Threshold*.
- **TellMostInformation($New? \in [True, False]$):** Tells whatever reveals the most information, whether this is the most information in total or the most new information.
- **TellDispensible:** Tells the next player with an unknown dispensible card the information needed to correctly identify that the card is dispensible. This rule will only target cards that can be identified to the holder as dispensible

with the addition of a single piece of information.

- **TellAnyoneAboutUsefulCard:** Tells the next player with a useful card either the remaining unknown suit of the card or the rank of the card.
- **TellAnyoneAboutUselessCard:** Tells the next player with a useless card either the remaining unknown suit of the card or the rank of the card.

A. Agents

1) *Legal Random:* This agent makes a move at random from the set of legal actions available to it at any given time step.

2) *Internal:* This is a clone of the agent presented by Osawa that shares the same name. It features memory of the information it has been told about its own hand but does not remember information about what other players have been told. The rules used in order are:

- PlaySafeCard
- OsawaDiscard
- TellPlayableCard
- TellRandomly
- DiscardRandomly

3) *Outer:* This is a clone of the agent presented by Osawa with the same name. It features knowledge of what the other agents have been told already, to avoid repeating Tell actions. The rules used in order are:

- PlaySafeCard
- OsawaDiscard
- TellPlayableCardOuter
- TellUnknown
- DiscardRandomly

4) *Cautious:* This is an agent derived from human game-play. The agent plays cautiously, never losing a life. The rules used in order are:

- PlayIfCertain
- PlaySafeCard
- TellAnyoneAboutUsefulCard
- OsawaDiscard
- DiscardRandomly

5) *IGGI*: This agent is a modification of Cautious. The alteration to a deterministic Discard function greatly aids the predictability of this player. The rules used in order are:

- PlayIfCertain
- PlaySafeCard
- TellAnyoneAboutUsefulCard
- OsawaDiscard
- DiscardOldestFirst

6) *Piers*: This is an agent designed to use IfRules to improve the overall score. Otherwise, it is similar to IGGI. The rules used in order are:

- IfRule (lives > 1 \wedge \neg deck.hasCardsLeft) Then (PlayProbablySafeCard(0.0))
- PlaySafeCard
- IfRule (lives > 1) Then (PlayProbablySafeCard(0.6))
- TellAnyoneAboutUsefulCard
- IfRule (information < 4) Then (TellDispensable)
- OsawaDiscard
- DiscardOldestFirst
- TellRandomly
- DiscardRandomly

The first IfRule is designed as a hail Mary in the end game: if there is nothing left to lose, try to gain a point. This derives from human play, when typically during the end game we make random plays if we know there is a playable card somewhere in our hand. This rule is more accurate, as it uses all the information it has gathered to calculate probabilities.

The second IfRule simply risks playing a card if there is a reasonable chance of its being safe.

The third IfRule is designed to try to provide more intelligent Tell conditions. If there is nothing useful to Tell and we are low on information, we set another agent up to be able to discard cards that are not needed. This means that the agents can burn through cards that are not helpful so as to try to obtain useful cards from the deck.

7) *Flawed*: This is an agent designed to be intelligent but with some flaws: it does not possess intelligent Tell rules, and has a risky Play rule as well. Understanding this agent is the key to playing well with it, because other agents can give it the information it needs to prevent it from playing poorly. The rules used in order are:

- PlaySafeCard
- PlayProbablySafeCard(0.25)
- TellRandomly
- OsawaDiscard
- DiscardOldestFirst
- DiscardRandomly

Giving information is the key to getting this agent to work intelligently. Without information, the intelligent rules can't fire, thereby leaving this agent to Tell randomly and Discard randomly — not a great strategy.

8) *Van den Bergh Rule*: This is the best rule-based agent from [18]. It was created by observing from human play that there are four main tasks:

- If I'm certain enough that a card is playable, Play it.
- If I'm certain enough that a card is useless, Discard it.
- Give a hint if possible.

- Discard a card.

Van den Bergh *et al* used a Genetic Algorithm (GA) to evolve the best options for each section, resulting in the following rules as an implementation:

- IfRule (lives > 1) Then (PlayProbablySafeCard(.6)) Else (PlaySafeCard)
- DiscardProbablyUselessCard(1.0)
- TellAnyoneAboutUsefulCard
- TellAnyoneAboutUselessCard
- TellMostInformation
- DiscardProbablyUselessCard(0.0)

9) *MCS*: This agent is a simple Monte Carlo Search (MCS) that uses a provided agent for the rollout phase. MCS is a technique that uses the Upper Confidence Bound (UCB) equation to select actions in a single step lookahead, with policy informed rollouts to evaluate those positions. It is essentially MCTS with a tree depth limit of one turn. In this paper, we name the agent MCS-[agent] to indicate which agent provided the rollout policy. For example, a MCS agent using IGGI as a policy would be named MCS-IGGI. The agent has a one-second time limit to return a move.

10) *IS-MCTS*: This agent uses a MCTS technique for handling games with partial observability as described in the paper by Cowling *et al* [19].

IS-MCTS is a modification to MCTS in which, on each iteration through the tree, the partially-observable game state is determined into a possible fully-observable state. This state remains consistent for the selection, expansion, rollout and backpropagation phases before being replaced by a new determination. The implementation uses a time limit for returning moves of one second per move.

11) *Predictor IS-MCTS*: This agent was provided with a copy of each of the agents that it was paired with to use in its prediction. The predicted agents were initialised with random seeds: this corresponds to the predictor's having knowledge of each agent's overall strategy but no knowledge of its internal workings.

The Predictor IS-MCTS agent modifies the selection, expansion and rollout phases of MCTS when considering nodes for other agents turns. The modifications remove Upper Confidence bound for Trees (UCT) for other agents' turns and replaces it with a query to the agent model to discover what that agent would do in that situation. The rollout phase is similarly modified. When making moves for its own turn, the predictor agent defaults to the legal random selection method used by IS-MCTS. The implementation maintains the one-second-per-move limit of IS-MCTS.

III. METHOD

A. Validation

In order first to validate our framework and AI implementations, we performed experiments using reimplementations of the Osawa and Van den Bergh agents. This involved recreating the experiments that they described in their papers and checking that we obtained similar results.

B. Full Test

The set of agents under test contained a mix of current research on *Hanabi* as well as some rule-based agents of our own. There is also a mix of strong and poor agents for balance. We tested all the agents from this list:

- Legal Random
- Outer
- IGGI
- Piers
- Flawed
- Van den Bergh Rule
- MCS-Legal Random
- MCS-IGGI
- MCS-Flawed
- IS-MCTS
- Predictor IS-MCTS

In each experiment, one of the agents was selected from the list above and the remaining agents were selected as a group from the list below. For example, in the first experiment the Legal Random agent would be alone among four IGGI agents — a concept we call *pairing*. The agents above were all paired in turn with:

- IGGI
- Internal
- Outer
- Legal Random
- Van den Bergh Rule
- Flawed
- Piers

200 random seeds were chosen, and for each seed every agent under test played two games with every agent with which it was paired. It did this for standard *Hanabi* rules with 2, 3, 4 and 5 players. Each agent under test played from a randomised position (first, second, third, fourth or fifth) determined by the seed. This ensured that each agent under test was in the same position for the same seed. Every agent therefore played $200(nSeeds) * 4(2, 3, 4 \text{ or } 5 \text{ Players}) * 7(nAgentPaired) * 2(reruns) = 11200$ games.

The configuration, final score and other basic state information were logged to a file upon completion of the game. The results were collated per agent and the mean score and number of turns taken were calculated. We also stored additional information about the final state of each game including the number of lives remaining and the information tokens remaining. When there are no lives remaining at the end of the game, this indicates that the players ran out of life tokens.

The full (human readable) game traces for each game are also stored, for evaluating agent behaviour and the effectiveness of strategies.

Finally, the configuration and results of each game are processed to obtain the mean score, mean number of moves per game and the mean remaining life and information tokens.

IV. RESULTS

A. Validation

The validation results are in Table I. The two Osawa agents obtained similar results in our system to those reported in

the original paper. The Van den Bergh Rule agent performed differently, appearing to be somewhat improved in our system.

B. Full Test

Table II shows the full results for this test. Predictor IS-MCTS outperformed IS-MCTS in this experiment, with an average score of 10.74 versus IS-MCTS's score of 5.9. MCS typically performed very similarly to the agent it was provided with for its rollouts; little benefit was apparent from using MCS with these agents over simply using their rules in the first place. Overall, Piers performed the best by a slim margin over MCS-IGGI, IGGI and Van den Bergh Rule. The Flawed agent was only a little better than Legal Random.

V. DISCUSSION

The Predictor IS-MCTS agent outperformed the IS-MCTS agent. This is mostly due to its better being able to take advantage of the effect of communication actions. As agents cannot see their own hands, the only way they gain information about their hands is via Tell actions; this then informs their decision process. When IS-MCTS appraises the moves of other agents in its tree, it considers all possible outcomes from that state. Some of these states will never occur in the real game because the paired agent would never select that action. The model that is available to Predictor IS-MCTS prunes the search to branches that are likely to occur in the game, resulting in more accurate statistics for the same number of iterations (Figure 2). The more deterministic the model, the lower the branching factor for the tree will be. Smaller branching factors concentrate the rollouts, resulting in potentially more accurate statistics regarding those positions. More accurate statistics should result in more intelligent game play.

The Predictor MCTS really shows its benefit with the Flawed agent as its partner. Table III shows each agent when paired with Flawed, with Predictor IS-MCTS in the clear lead ahead of other agents.

Interestingly, Predictor MCTS's poor overall score appears to come largely from two-player games, for which it scores significantly lower than usual. This can be explained by the decreased rollout length present in these games. The more players in the game, the fewer random moves will be made in the rollouts (selecting random moves tends to end games very quickly with low scores, as exemplified by Legal Random).

Table IV shows all the agents' average scores over each player count. Most agents tend to follow one of two trends: either performing better when there are more players in the game, or performing worse. Those that improve are typically poor players, with each new player added to the game on average being better than them. Those that decline are the opposite: more players added means more poorer players in the team. Predictor IS-MCTS isn't the only agent to exhibit trouble with two player games, with Outer experiencing some difficulty (despite having been designed for two-player games) and Van den Bergh Rule displaying a more prominent drop in performance. In 3, 4 and 5 player games, the Predictor IS-MCTS is the best player from the set of agents.

Agent	Our Average	Their Average	N Games	N Players
Internal	10.12 (SD 1.98)	10.97 (SD 1.94)	10^2	2
Outer	13.83 (SD 2.23)	14.53 (SD 2.24)	10^2	2
Van den Bergh Rule	16.95	15.4	10^4	3

TABLE I: Table of results of validation tests

Agent	Score (2.d.p)	Sem (2.d.p)
Piers	11.18	0.06
MCS-IGGI	10.97	0.06
IGGI	10.96	0.06
Van den Bergh Rule	10.88	0.06
Predictor IS-MCTS	10.74	0.06
Outer	10.2	0.05
IS-MCTS	5.9	0.04
MCS-Legal Random	5.45	0.04
MCS-Flawed	5.06	0.04
Flawed	5.02	0.04
Legal Random	4.59	0.04

TABLE II: Table of results with Score, Standard Error of the Mean and Ticks for each agent. Agents are sorted by score. N=11200

Agent	Score (2.d.p)	Sem (2.d.p)
Predictor IS-MCTS	4.82	0.06
IGGI	3.26	0.06
Piers	3.24	0.06
Van den Bergh Rule	3.23	0.06
MCS-IGGI	3.21	0.06
Outer	2.96	0.05
IS-MCTS	1.8	0.04
MCS-Legal Random	1.78	0.04
MCS-Flawed	1.67	0.04
Legal Random	1.65	0.04
Flawed	1.59	0.04

TABLE III: Table of results with Score, Standard Error of the Mean and Ticks for each agent paired with Flawed. Agents are sorted by score. N=1600

Agent	2	3	4	5
Flawed	3.52	4.69	5.43	6.45
IGGI	11.76	11.29	10.71	10.09
IS-MCTS	4.8	5.44	6.24	7.14
Legal Random	1.68	4.3	5.83	6.53
MCS-Flawed	3.61	4.72	5.43	6.48
MCS-IGGI	11.79	11.34	10.68	10.09
MCS-Legal Random	3.84	5.14	5.87	6.95
Outer	10.55	10.64	9.99	9.62
Piers	11.91	11.67	10.89	10.26
Predictor IS-MCTS	8.36	12.14	11.43	11.02
Van den Bergh Rule	10.55	11.76	10.91	10.29

TABLE IV: Average scores for each agent over 2, 3, 4 and 5 player games sorted alphabetically. Bold scores indicate the highest score for that column.

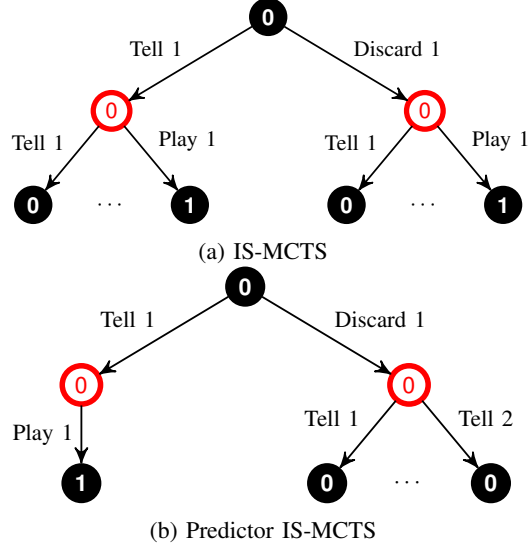


Fig. 2: Game trees from same state for both agents paired with Cautious illustrating the difference in tree size between IS-MCTS and Predictor IS-MCTS

VI. CONCLUSION

In conclusion, we found that agent modelling improves playing strength for tree search algorithms such as MCTS in the game of *Hanabi*. These results are consistent with the findings of [7].

VII. FUTURE WORK

There is a lot of scope for future work in this area. *Hanabi* has some additional variants in its rules that focus on the addition of a multi-coloured suit of cards. This suit also contains 3 1's, 2 2's, 3's and 4's as well as a single 5. The different variants are:

- Variant 1** Add the multi-coloured suit as a sixth suit to the game. Maximum score is boosted to 30.
- Variant 2** Same as Variant 1, but only a single tile of each number from the multi-coloured suit is added to the game.
- Variant 3** The multi-coloured suit now functions as a wild card in Tell actions, and cannot be directly called out. For example, if Player 1 tells Player 2 $\{(M, 2), (Y, 2), (B, 5), (B, 3)\}$ about all the blues, then cards 1, 3 and 4 will be indicated. With this setup, the multi-coloured cards can only be identified by contradicting information given, requiring 3 pieces of information to fully identify one.

Variant 1 would be simple to implement and test, but was omitted from this paper as being too off-topic. Variant 2 adds a little extra strategy, but is very similar to Variant 1. Variant 3 would require some additional work to implement, as well as appropriate modifications to the AI agents.

The Predictor IS-MCTS has a number of limitations that we aim to address. The agent requires access to an accurate model of the co-operators in advance. It would be better if the agent could instead attempt to learn agent strategies based on observations in the game state. This would lead naturally to a more complicated agent that started with a more generic capability but was able to build models of its team members and update those models as games go on. Testing how much information is needed to learn enough to significantly improve the scores that a team achieve would then need to be done.

VIII. ACKNOWLEDGEMENTS

This work was funded by the EPSRC Centre for Doctoral Training in Intelligent Games & Game Intelligence (IGGI) [EP/L015846/1]

The authors would like to thank members of the IGGI CDT for their assistance and support playing regular games of *Hanabi* with us.

REFERENCES

- [1] P. R. Williams, D. Perez-Liebana, and S. M. Lucas, "Cooperative games with partial observability."
- [2] J.-F. Baffier, M.-K. Chiu, Y. Diez, M. Korman, V. Mitsou, A. van Renssen, M. Roeloffzen, and Y. Uno, "Hanabi is np-complete, even for cheaters who look at their cards," *arXiv preprint arXiv:1603.01911*, 2016.
- [3] C. Baral, G. Gelfond, T. C. Son, and E. Pontelli, "Using answer set programming to model multi-agent scenarios involving agents' knowledge about other's knowledge," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, ser. AAMAS '10. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 259–266. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1838206.1838243>
- [4] A. Torreno, E. Onaindia, and O. Sapena, "Fmap: a heuristic approach to cooperative multi-agent planning," in *Proc. of DMAP Workshop of ICAPS*, vol. 13, 2013, pp. 84–92.
- [5] C. Geib, B. Craenen, and R. P. A. Petrick, "Generating collaborative behaviour through plan recognition and planning." [Online]. Available: <http://icaps16.icaps-conference.org/proceedings/dmap16.pdf#page=101>
- [6] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, 2005. [Online]. Available: <http://dx.doi.org/10.1007/s10458-005-2631-2>
- [7] S. Barrett, P. Stone, and S. Kraus, "Empirical evaluation of ad hoc teamwork in the pursuit domain," in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 567–574.
- [8] H. De Weerd, R. Verbrugge, and B. Verheij, "How much does it help to know what she knows you know? an agent-based simulation study," *Artificial Intelligence*, vol. 199, pp. 67–92, 2013.
- [9] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton *et al.*, "A Survey of Monte Carlo Tree Search Methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.
- [10] G. M. J. Chaslot, M. H. Winands, H. J. V. D. HERIK, J. W. Uiterwijk, and B. Bouzy, "Progressive strategies for monte-carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.
- [11] P. R. Williams, J. Walton-Rivers, D. Perez-Liebana, and S. M. Lucas, "Monte Carlo Tree Search Applied to Co-operative Problems," in *CEEC2015 - IEEE Conference on Computer Science and Electronic Engineering*, ser. IEEE CEEC. IEEE Computer Society, September 2015, pp. 219–224.
- [12] J. Walton-Rivers, "Controlling co-incidental non-player characters."
- [13] J. Rubin and I. Watson, "Computer poker: A review," *Artificial Intelligence*, vol. 175, no. 5, pp. 958–987, 2011.
- [14] D. Whitehouse, E. J. Powley, and P. I. Cowling, "Determinization and information set monte carlo tree search for the card game dou di zhu," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. IEEE, 2011, pp. 87–94.
- [15] H. Osawa, "Solving hanabi: Estimating hands by opponent's actions in cooperative game with incomplete information," in *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [16] C. Cox, J. De Silva, P. Deorsey, F. H. Kenter, T. Retter, and J. Tobin, "How to make the perfect fireworks display: Two strategies for hanabi," *Mathematics Magazine*, vol. 88, no. 5, pp. 323–336, 2015.
- [17] M. Van Den Bergh, F. S. M. I., and W. Kusters, "Hanabi, a co-operative game of fireworks," 2015.
- [18] M. J. van den Bergh, W. A. Kusters, and F. M. Spieksma, "Aspects of the cooperative card game hanabi."
- [19] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120–143, 2012.