UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

HENRIQUE BORGES MANZKE

Monte Carlo Tree Search with Transposition Detection and Probabilistic Inference for the Card Game of Hearts

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Science

Advisor: Prof. Dr. Anderson Tavares

Porto Alegre
January 2025

“Yeah, yeah, but your scientists were so preoccupied with whether or not they could that they didn’t stop to think if they should”

— Dr. Ian Malcolm

ABSTRACT

This work presents a Monte Carlo Tree Search (MCTS) algorithm for the imperfect information card game of Hearts. The algorithm introduces a novel mechanism specifically tailored to Hearts to detect transpositions of information sets, grouping nodes that, despite having different card distributions, follow the same general strategy. This significantly reduces computational redundancy. Additionally, it applies probabilistic inference to estimate the opponent's cards using historical data from thousands of self-play games, enabling more accurate predictions of hidden information. Experimental tests were conducted in a controlled environment to evaluate the impact of these enhancements on performance, and on the PlayOk website against experienced human players. The results demonstrate the algorithm's high-level performance, with potential for future applications in competitive AI for imperfect-information games.

Keywords: Artificial Intelligence. Monte Carlo Tree Search. Imperfect Information. Transpositions. Inference. Hearts.

Monte Carlo Tree Search com Detecção de Transposições e Inferência Probabilística para o Jogo de Cartas Copas

RESUMO

Este trabalho apresenta um algoritmo de Monte Carlo Tree Search (MCTS) para o jogo de cartas com informação imperfeita Hearts. O algoritmo introduz um mecanismo inovador, especificamente adaptado para Hearts, para detectar transposições de conjuntos de informação, agrupando nós que, apesar de possuírem distribuições de cartas diferentes, seguem a mesma estratégia geral. Isso reduz significativamente a redundância computacional. Além disso, aplica inferência probabilística para estimar as cartas dos oponentes usando dados históricos de milhares de jogos jogados contra si mesmo offline, permitindo previsões mais precisas sobre as informações ocultas. Testes experimentais foram conduzidos em um ambiente controlado para avaliar o impacto dessas melhorias no desempenho e no site PlayOk contra jogadores humanos experientes. Os resultados demonstram o desempenho em alto nível do algoritmo, com potencial para aplicações futuras em IA competitiva para jogos de informação imperfeita.

Palavras-chave: Inteligência Artificial. Busca de Monte Carlo em Árvores. Informação Imperfeita. Transposições. Inferencia. Copas.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Inteligence
DAG	Directed Acyclic Graph
ISMCTS	Information Set Monte Carlo Tree Search
SO-ISMCTS	Single-Observer Information Set Monte Carlo Tree Search
ISMCTS-T	Information Set Monte Carlo Tree Search with Transpositions
ISMCTS-TI	Information Set Monte Carlo Tree Search with Transpositions and Inference
MCTS	Monte Carlo Tree Search
PIMC	Perfect Information Monte Carlo
UCB	Upper Confidence Bound
UCT	Upper Confidence Bound for Trees

CONTENTS

1 INTRODUCTION

The game studied in this work is the trick-taking card game of Hearts. Unlike games with perfect information, such as Chess or Go, Hearts involves significant hidden information and uncertainty, making it computationally complex. No AI developed to date has reached expert-level performance in the game. This uncertainty necessitates advanced AI techniques capable of strategic reasoning under ambiguity.

A single game of Hearts can last more than 30 minutes, requiring four players to stay engaged throughout. This makes maintaining consistent participation challenging. In order for full games between humans to be played out, ranking systems are used to penalize players who abandoned the game by lowering their rankings. Nonetheless, forfeits remain frequent. As a result, human players are often matched against AI, allowing them to quit freely without prematurely ending the game for others. Therefore, strong Hearts AI agents are essential for providing challenging gameplay and enabling players to leave matches without disrupting others.

No AI developed for Hearts has matched or exceeded the performance of human players online. The agent by DEMİRDÖVER, Baykal and Alpaslan (2022) came the closest, achieving a 23.22% win rate. Developing AI agents for Hearts is challenging due to its large hidden information (up to 39 hidden cards) and the game's dual, conflicting goals: players must decide whether to try avoiding penalty points or to collect them all, depending on the round's context. Some prior works have ignored the game's hidden information by approximating it as a perfect information game (Sturtevant, 2008) (Bax, 2020), an approach with significant limitations (discussed in Subsection 2.3.2). Others have developed agents for hidden information gameplay but failed to achieve human-level performance (Whitehouse, 2014).

This study develops variants of the Information Set Monte Carlo Tree Search algorithm (ISMCTS) that explicitly handle the imperfect information nature of Hearts, enabling more robust decisions under uncertainty. The variants proposed are capable of identifying transpositions of states that are strategically identical, significantly reducing the reuse of computational resources and allowing deeper search. Furthermore, an inference method capable of guessing opponent cards with better than random accuracy is developed. It takes into account noticeable features of the opponent's behavior that strongly indicate their likely set of cards.

The agents were tested in a local environment against baseline versions lacking the

proposed enhancements. Thousands of game rounds were played, with card deals cycled between rounds to ensure fairness across agents. Performance metrics demonstrated the effectiveness of the proposed enhancements. Finally, the strongest proposed agent was tested against experienced human players online, showing it could compete at a high level.

Chapter 2 introduces the necessary game rules and technical background to understand the approaches proposed in this work. Chapter 3 goes over other works studying AI for playing Hearts and other imperfect information games. Chapter 4 presents a proposed algorithm capable of recognizing functionally identical information sets. Chapter 5 presents another proposed algorithm built on top of the transposition agent that can infer likely opponent hands. Chapter 6 examines both of the algorithms proposed to gather data on how well their mechanisms work and on their performance. Chapter 7 ends this work summarizing results and promoting future work.

2 BACKGROUND

This chapter introduces the necessary game rules and theoretical knowledge for understanding the algorithms used in the following chapters. Section 2.1 outlines the game rules of the Hearts variant studied. Section 2.2 introduces the MCTS family of search algorithms that acts as a foundation for all the following algorithms. Section 2.3 presents an MCTS variant tailored to games of imperfect information. Section 2.4 goes over modifications to MCTS that let it share information across equivalent states. Section 2.5 explains the concept of abstractions, a process that simplifies state representations by ignoring their irrelevant features. Finally section 2.6 examines inference in games, the prediction of hidden information, and its strategic counterpart, bluffing.

2.1 Hearts

Hearts is a popular imperfect-information trick-taking card game that originated in the 1880s. While the game has many variants, this study focuses on its most popular version, known as "Black Lady." Hearts is a multiplayer game typically played by 3 to 5 players, however, the standard and most commonly played format involves 4 players, which is the focus of this study.

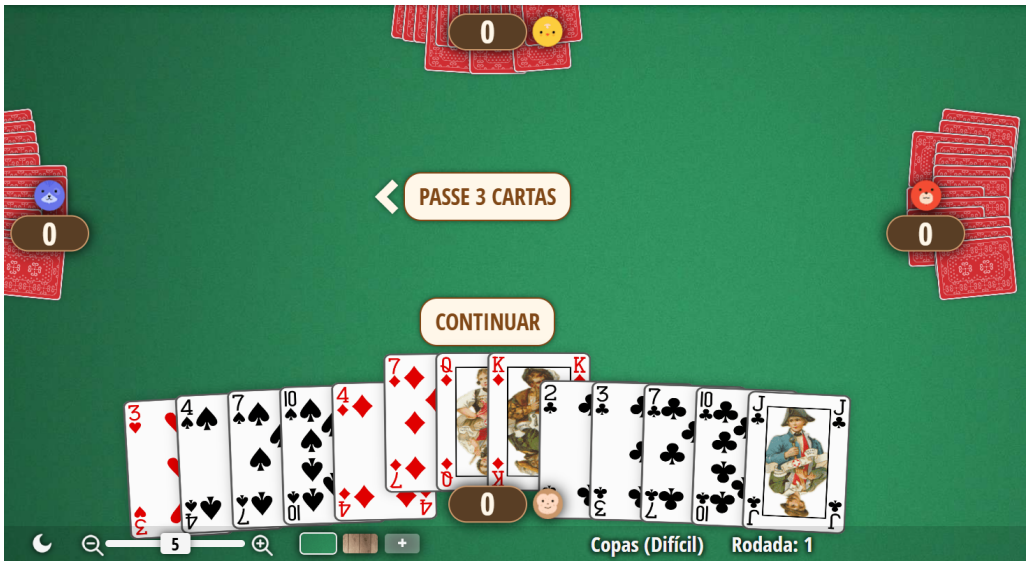
The game uses a standard deck of 52 playing cards without jokers. A full game is played through many rounds. At the start of each round, the deck is shuffled, and each player is dealt 13 cards. Each round consists of two phases: the passing phase and the playing phase.

2.1.1 Passing phase

During the passing phase, each player selects three cards from their hand and passes them face down to another player. Passing occurs simultaneously, the players cannot pick up the cards they have received prior to passing their own. The direction in which the cards are passed changes cyclically: in the first round, cards are passed to the player on the left; in the second, to the player across the table; in the third, to the player on the right; and in the fourth, no cards are passed (effectively skipping the passing of cards in this round). This cycle repeats every four rounds throughout the game. Figures

2.1 show a player passing and receiving cards in an actual round.

Figure 2.1 – A player selecting the three cards it will pass to the player to their left.

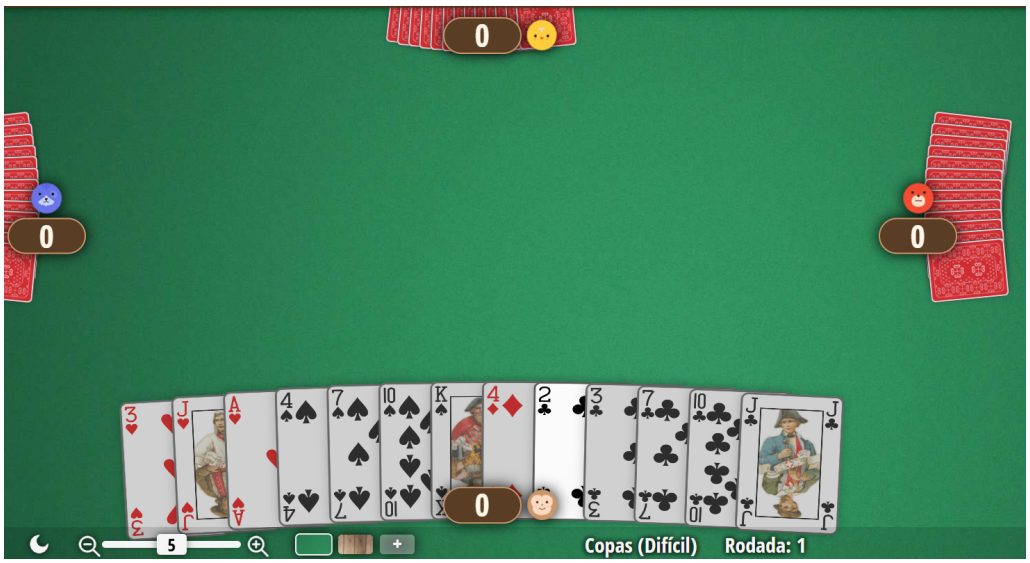


Source: Screenshot taken from *paciencia.co* (<https://www.paciencia.co/copas>).

2.1.2 Playing phase

This phase consists of tricks, a common mechanic in many card games. A trick begins when the leading player plays a leading card, followed by each of the other players playing a card in clockwise order. The winner of each trick is the one who plays the highest-ranking card from the leading suit. Cards are ranked from lowest to highest as follows: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, and A. The winner takes all cards in the trick and leads the next one. This process continues until all cards have been played. Since each player plays one card per trick and there are 13 cards per player, the playing phase always lasts 13 tricks. To initiate a new round, the player holding the 2 ♣ must play it as the first leading card, as shown in Figure 2.2.

Figure 2.2 – The player with the 2 ♣ having to lead the first trick with it.



Source: Screenshot taken from *paciencia.co* (<https://www.paciencia.co/copas>).

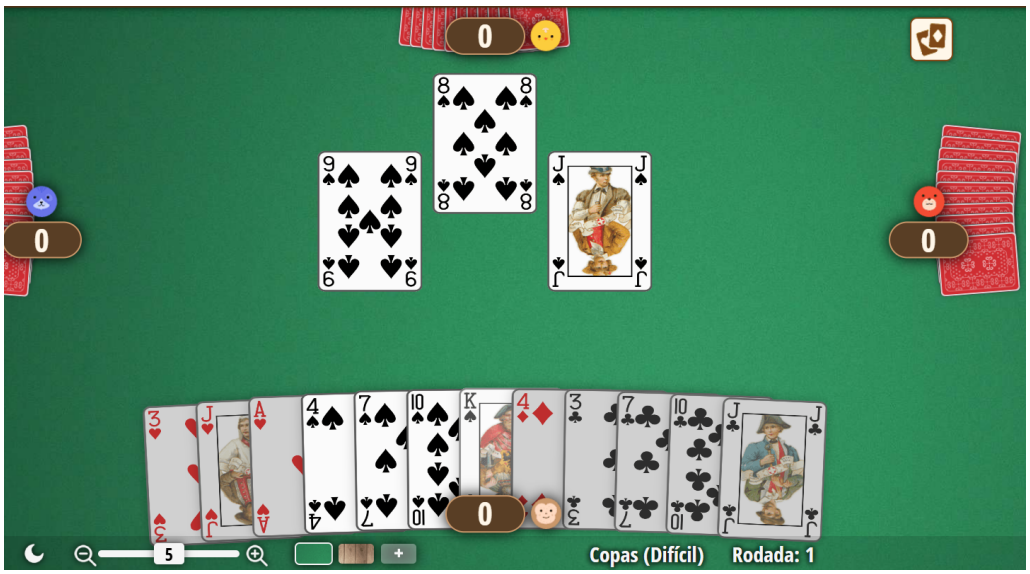
Certain cards are worth points when taken. Every card in the suit of Hearts is worth 1 and the Q♠ is worth 13, for a total of 26 points per round. All other cards are not worth any points.

There are a few rules restricting which cards from a hand can be played:

- A player cannot play cards worth points during the first trick.
- A player must follow the suit of the leading card if possible.
- If they are the ones leading, their only restriction is that they cannot lead with Hearts until the suit of Hearts has been "broken". Hearts are considered broken if a Heart has been played in a previous trick.

These rules do not apply if a player has no other option but to break them. Figure 2.3 shows a round in the playing phase.

Figure 2.3 – The player's turn on the second trick of a round. The left player led the trick, and in clockwise order, the front and right players played their cards. The player must play a card of the same suit as the lead.



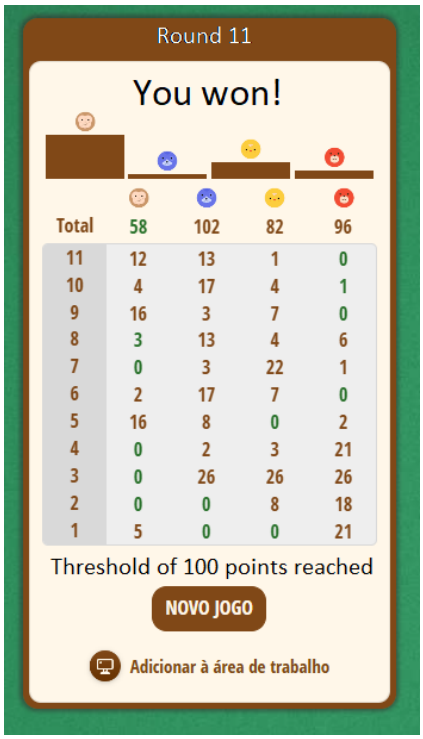
Source: Screenshot taken from *paciencia.co* (<https://www.paciencia.co/copas>).

The order in which cards are played during the playing phase ultimately determines the outcome of rounds. The passing of cards can only do so much, as it is limited to discarding only 3 of the 13 cards in the hand. So while the passing phase is informative and influences the coming round. It is not the primary focus of this work and will only receive minor attention.

At the end of a round, each player scores points equal to the total value of the cards they took. If a player successfully took all 26 points (all ♡ cards and the Q♠), it is referred to as "shooting the moon". In this case, the scoring is adjusted: the player who "shot the moon" receives 0 points, while all other players are each penalized with 26 points. Attempting to shoot the moon and failing is devastating: the player may end up collecting almost all possible points. Consequently, while shooting the moon is the best outcome for the player, it also requires long-term planning to ensure it's worth trying ahead of time. Thus, for each round, players face two conflicting strategies: either to minimize the points taken or to capture all of them to shoot the moon. Finally, the points scored in the round are added to each player's total game score.

The objective of the game is to have the fewest points when the game ends. The game concludes when one of the players reaches or exceeds a predetermined score threshold. An example of a game played to a score threshold of 100 points is shown in Figure 2.4.

Figure 2.4 – The final result of a Hearts game played among 4 players for 11 rounds until the blue player crossed the 100 points threshold, ending the game. The first player, having the fewest points, was declared the winner.



Source: Screenshot taken from *paciencia.co* (<https://www.paciencia.co/copas>) and partially translated by the author.

2.2 MCTS

This section presents the Monte Carlo Tree Search (MCTS) family of algorithms, which serve as the foundation for the algorithms analyzed in this study. MCTS is a powerful general-purpose AI technique that can efficiently search games with large state spaces with no domain knowledge. It has attained success in various domains, most notably in deterministic games of perfect information such as chess and Go (Silver et al., 2016), but also in general game playing (Finnsson, 2007).

It iteratively builds a search tree guided by the results of simulations. Simulation paths that look more promising are explored more often, making it ideal for games with large search spaces. Each iteration consists of four steps: selection, expansion, simulation and backpropagation as shown in Figure 2.5.

Selection: Starts at the root node and uses a tree policy to select the next child node. The most common tree policy is the upper-confidence bound (UCB) algorithm (Auer, 2002) applied to trees (UCT) (Kocsis; Szepesvári, 2006), which treats the selection of actions down the tree as a multi-armed bandit problem. The UCT algorithm uses Equation 2.1 to measure how appropriate it is to go from a node v to one of its child nodes v'. $Q(x)$ represents the average reward of a node x, $N(x)$ is the number of times a node x was visited and c is the exploration constant that balances exploitation versus exploration. The UCT policy selects the child node v' with the highest $UCT(v, v')$ value.

$$\text{UCT}(v, v') = Q(v') + c\sqrt{\frac{\ln(N(v))}{N(v')}} \tag{2.1}$$

This step ends once it reaches a node that is either terminal, or that has not been fully explored. A node not being fully explored means one of its available actions leads to a node that has not been added to the tree yet. If the node is terminal, the algorithm jumps directly to the backpropagation step, otherwise it moves to the expansion step.

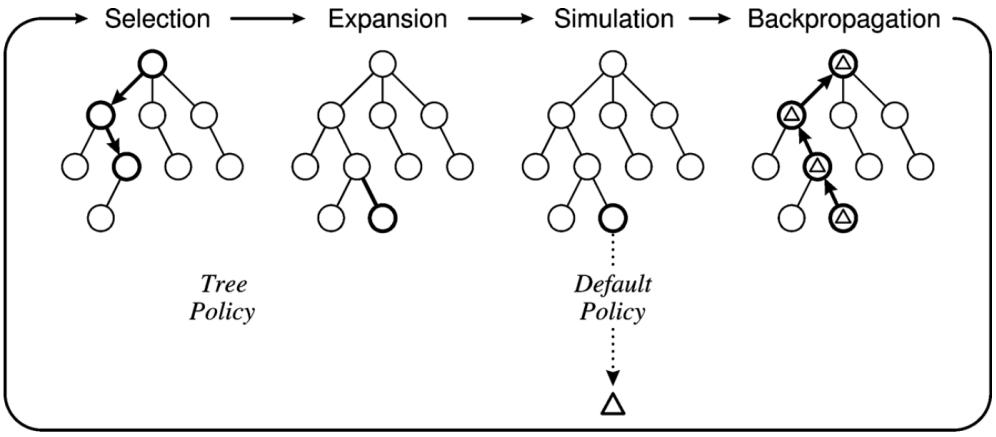
Expansion: Creates the missing child node from the selection step and moves to the simulation step.

Simulation: Simulates a possible outcome by playing randomly or heuristically from the newly created node until reaching a terminal state. Then the terminal state is evaluated with a reward function and the algorithm moves on to the backpropagation step.

Backpropagation: Updates the values of all nodes along the path from the newly

created node back to the root: the number of visits $N(s, a)$ is increased by one and $Q(s, a)$ is updated with the reward from the simulation.

Figure 2.5 – The MCTS 4-step iteration cycle



Source: (Browne et al., 2012)

After the predefined number of iterations is run, or time has run out, the algorithm returns the root action that received the most visits. MCTS is an anytime algorithm, meaning it can provide a valid solution regardless of when its search is stopped.

For more details, refer to the survey by Browne et al. (2012).

2.3 Information Set MCTS (ISMCTS)

Information Set MCTS was first introduced by Cowling, Powley and Whitehouse (2012) as a new family of MCTS algorithms adapted to games with hidden information, technically known as imperfect information games. This section will first provide the necessary knowledge to understand imperfect information games and lastly describe the algorithm.

2.3.1 Information Sets

Information sets are sets of possible states that the observing player cannot distinguish with the available knowledge. The partitioning of states into information sets is specific to each player, but not necessarily unique, as players may share the same knowledge. In Hearts, the players often do not know their opponents' hands, so the information

set they observe is the set of all possible permutations of the opponents' cards. In games of perfect information, there is only one possible state in each information set as there is no hidden information.

In Hearts, players start with access to just their private information, but as the round progresses, information is increasingly revealed, diminishing the player's uncertainty. This means that information sets grow smaller during a round, eventually containing a single possible state. The rate at which the number of possible states inside an information set shrinks is domain specific and is called the disambiguation factor (*df*). The term was coined in (Long et al., 2010) as an attempt to understand the success of the Perfect Information Monte Carlo (PIMC) (presented in subsection 2.3.2) in certain games of imperfect information.

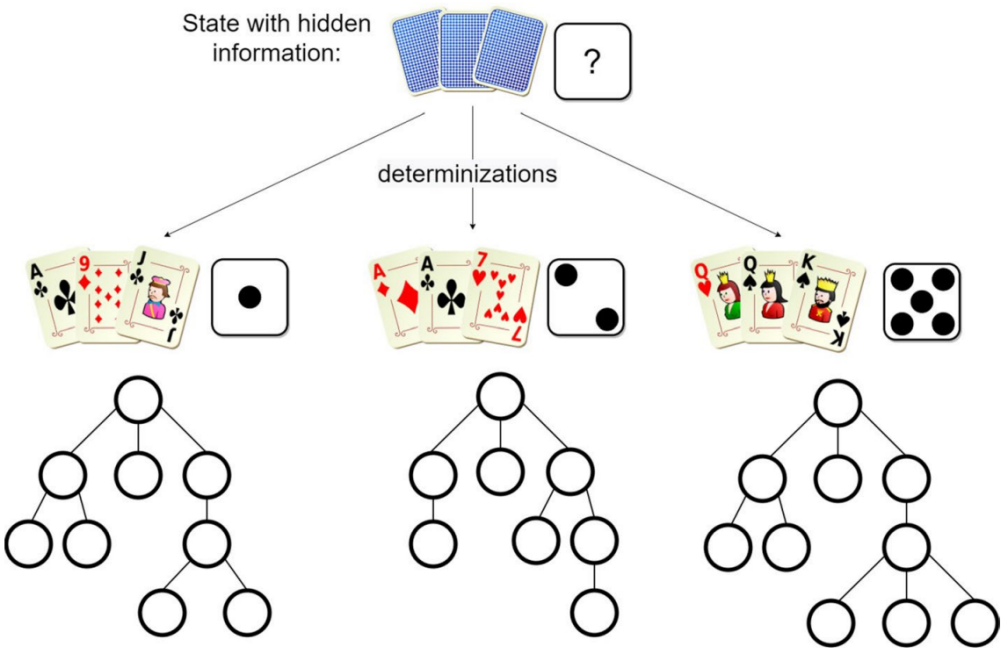
When the observer of an information set is the player about to act, the set of actions available will always be the same. This is a result of information sets being sets of totally indistinguishable states to the observing player, including by their action sets. However, if it's the opponent's turn to act, the exact set of actions available to them may be unknown to the observing player.

2.3.2 Determinization and Perfect Information Monte Carlo (PIMC)

One common approach to handling imperfect information in games is to relax the game into one of perfect information. This is done by sampling possible states from the player's information set and analyzing those samples as in a perfect information game. To sample states, all the feature values unknown to the player are determinized (guessed), resulting in what is known as a determinization. The advantage of determinizations is that they can be analyzed with many well-established perfect information algorithms. An example of this process is shown in Figure 2.6.

The Perfect Information Monte Carlo (PIMC) algorithm predates MCTS and uses Monte Carlo sampling of determinizations to play imperfect information games. At the root node of the search, a possible state s from the information set is sampled, a perfect information algorithm is used to analyze it and the results are stored. This repeats until a sufficient sample size has been analyzed. PIMC then either selects the action that was the best on average or the action that was the best the most times. It has achieved expert level play in games like Bridge (Ginsberg, 2001) and Skat (Buro et al., 2009) and strong play in Hearts (Sturtevant, 2008).

Figure 2.6 – An example of determinization applied to a game where the hidden information consists of the opponent's three cards and the dice they have rolled. Several possible determinizations are sampled from the information set, each resulting in its own game tree for analysis.



Source: (Świechowski et al., 2023)

However, there are many drawbacks with this approach. First, given that it treats the game as a perfect information game, it never tries to gather or hide information. Second, due to all determinizations being compatible with the searching player's knowledge, its opponent model makes decisions with perfect clairvoyance of its knowledge, even though opponents do not necessarily have access to it. Third, the computational budget available is split between every determinization sampled, so there is a tradeoff between using a good sample size and how much of the budget each sample gets. Two other problems are identified by (Frank; Basin, 1998):

- Strategy fusion: as each determinization is analyzed in isolation, the algorithm picks actions that are good in that particular perfect information scenario, not concerning itself with how good that action fairs in other scenarios. For example, if PIMC were to analyze a problem of whether to buy a lottery ticket or not, with the hidden information being the winning number, it would always randomly buy a lottery ticket thinking its reward is guaranteed, and never pick the more financially advisable option of not buying a ticket at all.
- Non-locality: Since players are attempting to direct the game towards states that

they consider more beneficial, such states have a higher likelihood of occurring and thus should be sampled more often. Therefore, considering the current (local) root node without accounting for the action history is not guaranteed to return the best possible action.

The degree to which these flaws affect the algorithm's performance depends on the domain, sometimes having a negligible negative impact. Such differences in performance across games, including Hearts, have been studied by (Long et al., 2010). Hearts seems to be only slightly affected by these problems.

2.3.3 The ISMCTS algorithm

Information Set Monte Carlo Tree Search (ISMCTS) is a family of MCTS algorithms where nodes represent information sets rather than states and all determinizations share the same search tree, as opposed to PIMC (Section 2.3.2), where each determinization builds its own tree. This is possible by mapping the determinized states to their corresponding information sets from the player's perspective. ISMCTS spares computation resources, and the algorithm picks an action at the root that is good in general across all determinizations, solving PIMC's problem of strategy fusion.

ISMCTS considers a different determinization s for each iteration. While the same tree is used, the sets of available actions at the opponents' nodes may change depending on s. This introduces an issue: when an opponent node v is visited multiple times by different determinizations, some of its actions will be available more times than others. Since this is just like a multiarmed bandit problem, except that only a subset of the actions are available on each trial, it is referred to by Cowling, Powley and Whitehouse (2012) as a *subset-armed bandit* problem. This difference would lead standard bandit algorithms to over-explore the rarer actions, as their UCB value would be much higher, and take focus away from exploiting actions that are available more often. The solution proposed at (Cowling; Powley; Whitehouse, 2012) is to slightly modify standard bandit algorithms: the number of times a node v was visited $N(v)$ is replaced by $B(v')$, the number of times child node v' was available to be selected. The modified UCT tree policy is shown in Equation 2.2.

$$\text{UCT}(v, v') = Q(v') + c\sqrt{\frac{\ln(B(v'))}{N(v')}} \tag{2.2}$$

The specific variant of ISMCTS used in this work is called single-observer information

set MCTS (SO-ISMCTS). It faces issues when games have partially observed actions, but as all actions are fully observable during the playing phase of Hearts, using this variant is sufficient for that phase of the game. Pseudocode of SO-ISMCTS is presented in Algorithm 1. There is no specific bandit algorithm that should be used during selection, but UCB modified for subset-armed bandits as described by Equation 2.2 is the standard, used by the authors for all of their experiments. For simplicity, from this point forward, we will refer to SO-ISMCTS as ISMCTS, as it is the only ISMCTS variant that will be used.

Algorithm 1 Pseudocode for the SO-ISMCTS algorithm

```
1:  function SO-ISMCTS(I, n)
2:      Create a tree with root v_0 representing information set I
3:      while stopping criteria not met do
4:          Sample a determinization s at random from I
5:
6:          // Selection
7:          repeat
8:              Choose a child node v (restricted to actions compatible with s) using the
    chosen bandit algorithm
9:          until v is terminal or some action from v leads to an information set not cur-
    rently in the tree
10:
11:         // Expansion
12:         if the v is nonterminal then
13:             Randomly choose an action a from v compatible with s leading to an
    information set v' not on the tree
14:             Add v' as a child node of v
15:         end if
16:
17:         // Simulation
18:         Run a simulation from v' until a terminal node
19:
20:         // Backpropagation
21:         for each node visited during this iteration do
22:             Update the node's visit count and total simulation reward
23:             Update the availability count of the node and its available siblings
24:         end for
25:     end while
        return an action a at root v_0 with the most visits
26: end function
```

2.4 Transpositions in MCTS

In traditional MCTS, the search space is treated as a tree. When distinct sequences of actions starting from the root lead to the same states, those states are classified as transpositions. If transpositions are merged and handled as a single node, the search structure becomes a directed acyclic graph (DAG) rather than a tree. This reduces redundancy and improves computational efficiency. As the round of Hearts always ends after the 52 cards were played, there can be no cycles, and the game tree can be represented as a DAG. However, transpositions only occur if the exact history of actions is ignored, this leads to a loss of knowledge useful for inference that may need to be compensated for.

In order to better adapt MCTS to explore a DAG, this work follows advice from Saffidine, Cazenave and Méhat (2012), which recommends to store aggregated payoff values on edges instead of nodes. On a tree, there would be a one-to-one correspondence between nodes and actions, and this change would make no difference. But, as we will be working with a DAG, nodes can have multiple parent and child nodes, and extracting the average reward of an action becomes less straightforward. This modified UCT algorithm uses Equation 2.3 to measure how appropriate it is to select edge e. Edge e connects a node v to one of its child nodes v'. $Q(e)$ represents the average reward of edge e, $N(e)$ is the number of times edge e was taken, $B(e)$ is the number of times edge e was available and c is the exploration constant that balances exploitation versus exploration. The UCT policy selects the edge e with the highest $UCT(e)$ value.

$$\text{UCT}(e) = Q(e) + c\sqrt{\frac{\ln(B(e))}{N(e)}} \tag{2.3}$$

States are stored in a transposition table for efficient lookup. If each entry on the table holds the necessary information such as visits, availability, and child nodes we can fully discard the old tree structure, storing all node and edge data directly in the table.

2.5 State Abstraction

Abstraction is the process of simplifying the representation of states while preserving relevant information necessary for decision-making. When states are abstracted, there is a greater chance of finding transpositions, since there are fewer remaining features in states that can set them apart. Therefore, abstraction is often used to group together func-

tionally similar states, reducing the complexity of a problem. There are 2 types of state abstraction:

1. **Lossless abstraction**: No relevant information is lost. For example, in tic-tac-toe, boards that are rotations or reflections of one another can be considered the same.
2. **Lossy abstraction**: Some information that may impact decision-making is lost. An example taken from (Brown; Sandholm; Machine, 2017): grouping bets of similar sizes in Poker into a single category. Bets of 500$ and 501$ can be considered near identical at little cost.

In this work, the state abstractions used for detecting transpositions are lossless.

2.6 Inference and bluffing

A belief distribution determines the probability of each state in an information set being the real state. Inference is described in the field of Game AI as updating the belief distribution in response to another player's observed actions (Cowling; Whitehouse; Powley, 2015). The more players rely on inference, the more incentive there is for bluffing. Bluffing can be considered the same as attempting to keep your private information hidden or mislead your opponents into believing something specific by taking actions that would be sub-optimal in a perfect information version of the game.

The game of Skat is another imperfect information trick-taking card game with a very large search space. In Buro et al. (2009) developed a Skat agent that infers the likelihood of a possible state by observing data collected offline. Their work makes this possible by collecting data, not for each state, as there are too many, but from high-level features that can be found in states. The estimate of the probability of a state given the observed move is calculated by:

$$P(\text{state} \mid \text{move}) = \prod_i P(f_i)P(f_i \mid \text{move}) \ , \tag{2.4}$$

where $P(f_i|\text{move})$ is the chance of one of the state features (f_i) being present given an observed move $move$, this value comes from a dataset look-up. $P(f_i)$ is the chance of a feature appearing in a possible state and can be calculated at runtime. We will use a similar approach to apply inference to Hearts in Chapter 5.

3 RELATED WORKS

There are few modern works directed towards improving AI play in the full game of imperfect information Hearts. More popular imperfect information games such as Poker (Brown; Sandholm; Machine, 2017), Skat (Germany's national card game) (Rebstock et al., 2019) and Scrabble (Richards; Amir, 2007) have received more attention. Some prior studies have limited themselves to creating an agent for a simplified variant of Hearts (Sturtevant; White, 2007), (Bax, 2020). All studies have failed to surpass the human expert level of performance.

Ginsberg (2001) describes how the strongest Bridge playing program at the time was developed. Their techniques included PIMC and partition search. Partition search can be applied in games where outcomes depend on local features (such as checkmating the king in chess or aligning three X's or O's in tic-tac-toe). The technique stores states that conclude similarly in a transposition table, improving computational efficiency by avoiding redundant evaluations. Partition search was used to more efficiently analyze a perfect information version of Bridge. It uses transposition tables to store similar states and can be applied to any two-player constant-sum game.

One of the initial studies on Hearts was conducted by Sturtevant and White (2007). The authors developed a player for a simplified version of Hearts, without restrictions on card play and no passing phase using stochastic linear regression and TD learning. By exhaustively combining basic game features into more expressive representations, they trained their player through self-play and against search-based opponents. Their learned player outperformed one of the best-known Hearts programs at the time.

Sturtevant (2008) analyzes how well UCT performs in multiplayer games, including in the perfect information version of Hearts. The study concludes that UCT outperforms previously existing programs in Hearts. However, the UCT enhancements tested did not improve the quality of play as the number of simulations increased, instead, they caused performance to deteriorate. The authors proposed 2 possible explanations: First, since all cards will eventually be played, the key factor is whether a card is played in the right moment. This implies that enhancements ignoring context will fail. Second, Hearts has a small branching factor (70% of the time there are two or fewer actions), making playout policies less critical for guiding the simulations.

While earlier studies treated Hearts as a game of perfect information, Whitehouse (2014) developed an ISMCTS algorithm to explicitly handle the imperfect information

of Hearts. The winner of a Hearts game is decided once a player surpasses the score threshold, therefore, to evaluate terminal states, randomly sampled round outcomes were added to each player's score until the threshold was surpassed. This approach resembles simulating a game to its end, but is much more efficient due to utilizing precomputed outcomes instead of relying on real-time calculations. The round outcomes to be added were sampled from an offline dataset of 10,000 historical round outcomes, reached during previous rounds. However, this method introduces an issue: small differences in terminal scores become harder to discern due to the influence of randomness in the selection process, potentially reducing the accuracy of evaluations. Nonetheless, it was the first step towards implementing dynamic rewards that account for the current score context when evaluating terminal states. It performed similarly to the AI in the Microsoft version, with a win rate of approximately 25% against it.

Powley, Cowling and Whitehouse (2014) introduces the Information Capture And ReUse Strategy (ICARUS) framework. This framework describes and combines information-sharing enhancements to MCTS. Among these, the EPisodic Information Capture and Reuse (EPIC) enhancement is particularly relevant to this work. The authors applied EPIC to Hearts and other games of imperfect information. It divides the search space into episodic time windows the authors refer to as episodes. States that correspond to the same position in different episodes share information, independently of the game context where the episodes happened. In Hearts, each trick was treated as an episode that may occur at different times with similar results.

The authors who first published the ISMCTS family of algorithms made a follow-up study (Cowling; Whitehouse; Powley, 2015). In it, modifications to ISMCTS make it capable of inference and bluffing with very promising results. Unfortunately, it can only be applied to games with a very small number of information sets, such as The Resistance, which was the subject of study in their work.

Bax (2020) tests how the quality of the state sampling method affects the performance of the PIMC algorithm in the Dutch variant of Hearts, which uses 32 cards and has fewer restrictions than the standard version. It limits itself to comparing methods that may sample illegal states against a method that never does so, and does not try to improve the sampling rates of states according to their believed likelihood.

A recent study by DEMİRDÖVER, Baykal and Alpaslan (2022) proposes a self-learning Hearts agent that employs a collaborative set of subagents specialized in different aspects of the game. To deal with the contradictory goals that exist in Hearts, a group of

subagents called “decision agents” determines the playstyle, selecting between normal, shooting the moon and preventing an opponent from shooting. Each playstyle has corresponding subagents that can dynamically switch during runtime. The study reports achieving the best performance against human players to date: on matches of a human against 3 of its AIs, each AI had a 22.67% win rate, and a 23.33% win rate when not trying to prevent players from shooting the moon. However the agent has not surpassed human performance. One notable limitation is its inability to alternate between normal and moonshot playstyles after a round begins: its first decision agent decides whether to try to shoot the moon at the start of a round and cannot change that decision later on as unforeseen circumstances arise.

Schlagnitweit (2023) uses inference to filter what states are sampled for a Monte Carlo agent that plays the imperfect information card game of Schnapsen [1]. After observing an opponent move, all possible opponent hands are visited and analyzed by a predefined policy. Only states where the policy confidently picked the same move that was observed are sampled.

[1] A description of the game Schnapsen can be found at <https://en.wikipedia.org/wiki/Schnapsen>.

4 TRANSPOSITIONS AGENT

This chapter describes the proposed method developed in this work for identifying transpositions of strategically identical information sets in Hearts and applying how it is applied to ISMCTS. Across some of the following sections, the game's rules will be mentioned frequently, refer back to Chapter 2.1 if needed.

4.1 Information set abstraction

In the following subsections, we'll go over all the types of information that need to be kept when developing a strategy for an information set in Hearts, and how they are abstracted.

4.1.1 Trick abstraction

For every trick, we keep knowledge of who lead it, what card is currently winning it and how many points it's worth.

4.1.2 Points taken abstraction

When a player takes all points in a round, they "shoot the moon" and finish the round with the best possible reward. Once a player has taken points, shooting the moon becomes, by this definition, impossible for other players. Subsequently, when two or more players have taken points, shooting the moon becomes impossible for all players. Thus, while the exact amount of points each player already has taken does not affect the playing strategy, which players have taken points does. Therefore, the amount of points each of the 4 players took can be abstracted to one of six scenarios:

1. No one has taken points yet, and all can shoot the moon.
2. The first player has taken points, and only they can shoot the moon.
3. The second player has taken points, and only they can shoot the moon.
4. The third player has taken points, and only they can shoot the moon.
5. The fourth player has taken points, and only they can shoot the moon.

6. More than one player have taken points, and no one can shoot the moon.

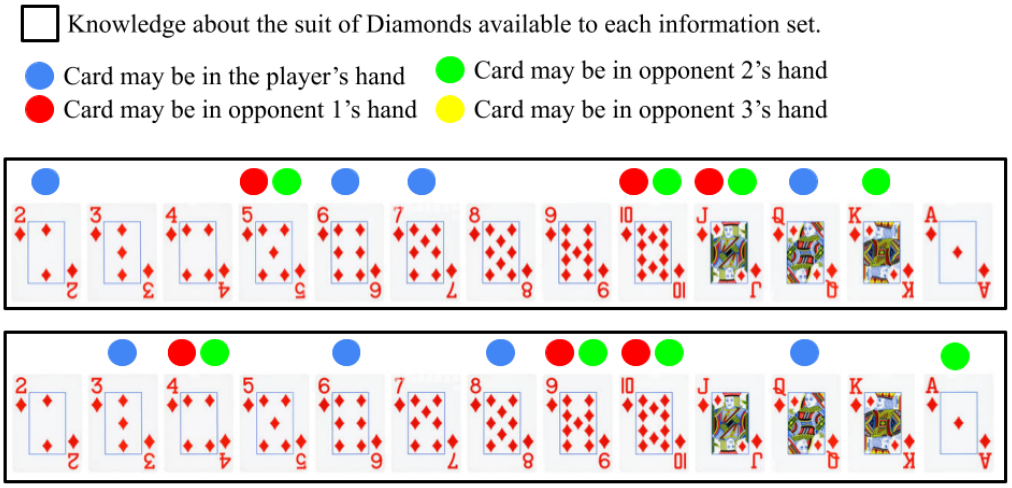
4.1.3 Card abstraction

Card ranks are used to define the ordering of cards in a suit, sorting them from lowest to highest ranking. However, as long as the relative order of cards is known, the exact rank of cards does not need to be remembered. The exception is the rank of the Q♠, as its rank carries the special property of being worth 13 points, and it is not used simply to determine the card's order. Only 4 properties need to be known to define a card without using its rank:

- Its suit.
- Who might be holding it.
- Its relative order among the remaining cards in its suit.
- If it is the Q♠.

There may be more concise sets of defining properties that also abstract the card's suit, however the properties mentioned seemed easier to visualize and implement. Figure 4.1 shows the knowledge two different information sets have of the Diamonds suit. If card rank abstraction is applied to them, we get the same result for both: The player may hold the 1st, 3rd, 4th and 7th cards of Diamonds, opponent 1 may hold the 2nd, 5th and 6th cards of Diamonds, opponent 2 may hold the 2nd, 5th, 6th and 7th cards of Diamonds and opponent 3 has no cards of Diamonds.

Figure 4.1 – Two examples of what information sets could know about cards of the suit of Diamonds.

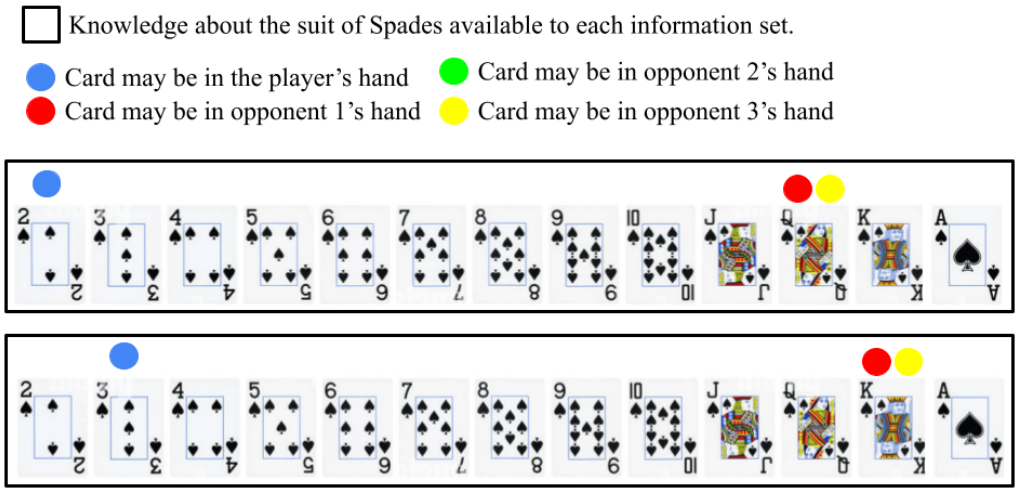


Source: Author

Now, let's explore a scenario where even after card rank abstraction, two information sets remain different. Figure 4.2 shows the knowledge two different information sets have of the Spades suit. If card rank abstraction is applied to them, we get different results for each:

- **Upper information set:** The 1st card of Spades may be with the player, the 2nd card of Spades is the Q♠ and may be held by opponent's 1 or 3.
- **Lower information set:** The 1st card of Spades may be with the player, the 2nd card of Spades may be held by opponent's 1 or 3.

Figure 4.2 – Two examples of what information sets could know about cards of the suit of Spades.



Source: Author

4.1.4 Missing information that can be extracted from the abstraction

Some relevant information may appear to be lost after abstraction, but they can be extracted from the final result:

- The current trick's number can be known by observing the number of cards played. $currentTrick = (numberOfCardsPlayed)/4 + 1$
- Whose turn it is to act can be known by a using the number of cards played and who is leading the current trick. $whoseTurnItIs = (leadingPlayer + numberOfCardsPlayed)\%4 + 1$
- The leading suit is the suit of the current winning card.
- To know if Hearts is broken, see if any Hearts cards have been played.

4.2 Reward function

Before introducing the proposed ISMCTS variant capable of handling abstractions, it is important to explain the reward function used by this and other MCTS algorithms involved in this work. In Hearts, a round reaches a terminal state when all the cards worth points have been taken and the outcome is decided. As a result, it is common for the search to end prior to the 13th trick. Once a terminal node is reached, we get the final

number of points each player took. However, evaluating the value of the final result is not straightforward. If a player were to simply try minimizing the number of points they take, they will see no difference between taking no points and shooting the moon, even though the number of points all their opponents took is 3 times greater in the latter case. And if the player was trying to maximize the number of points their opponents took, they would actively help the opponents shoot the moon, as that would also give their other two opponents 26 points each. Therefore, a good reward function must balance the goals of minimizing the player's points and maximizing the points taken by opponents. Thus, the reward function chosen is the following:

$$R(s,i) = \sum_{j \neq i} (P(s,j) - P(s,i)) \ , \tag{4.1}$$

where $R(s,i)$ is the reward player i gets for reaching terminal state s and $P(s,p)$ is the number of points player p gets for reaching state s. The reward represents how much the player has improved or worsened its placement relative to other players, so it always sums to 0. Table 4.1 shows a few examples of calculated rewards. The third line presents a terminal state where player 1 shot the moon.

Table 4.1 – Examples of rewards each player receives in different hypothetical terminal states

State	*P(s, 1)*	*P(s, 2)*	*P(s, 3)*	*P(s, 4)*	*R(s, 1)*	*R(s, 2)*	*R(s, 3)*	*R(s, 4)*
State 1	13	2	7	4	-26	18	-2	10
State 2	0	6	7	13	26	2	-2	-26
State 3	0	26	26	26	72	-26	-26	-26
State 4	0	0	25	1	26	26	-74	22
State 5	13	13	0	0	-26	-26	26	26

Source: Author

4.3 ISMCTS-T

In this work, an ISMCTS variant capable of recognizing transpositions is developed. We will refer to it as ISMCTS with Transpositions (ISMCTS-T) Its nodes store abstracted information sets using the representation defined in Section 4.1. The nodes are stored on a transposition table. Each node holds a list of edges leading to their child nodes. The edges hold the necessary information for selection: visit count, availability count and aggregated reward.

The selection and backpropagation methods used in this work are the standard ones of ISMCTS from Algorithm 1, meaning no modifications were made to leverage information indirectly available and only actions along the descent path are updated with the payoff (called *update-descent*). As nodes have multiple parents, the descent path has to be stored during the iteration for backtracking to be later possible.

While traditional ISMCTS samples new determinizations for every iteration, to improve efficiency, our algorithm only samples a fixed set of k determinizations to use for the whole search. This imposes a tradeoff: higher k is more costly, but allows more determinizations, favoring a general strategy to be chosen near the root node.

As new information is revealed, most determinizations from the previous search become inconsistent with the actual state. Therefore, at the start of each search, a new set of k determinizations consistent with the revealed information must be sampled. Additionally, the search graph cannot be reused and must be reset at the beginning of each search, since older graphs retain values based on outdated determinizations.

The algorithm will be referred to on later chapters as ISMCTS-T. Its pseudocode is presented by Algorithm 2.

Algorithm 2 Pseudocode for ISMCTS-T

function ISMCTS-TP(I, n)
    Initialize the transposition table
    Sample k determinizations and store them
    Create a root node v_0 representing information set I
    Add v_0 to the transposition table
    for n iterations **do**
        Pick a determinization s at random from the k sampled

        // Selection
        repeat
            if at node v an action a leads to a node v' on the transposition table, but the correspondent edge does not exist **then**
                Create the missing edge.
            end if
            Descend one node in the tree (restricted to actions compatible with s) using UCT modified for the subset-armed bandit problem.
        until v is terminal or some action from v leads to a node not in the transposition table

        // Expansion
        if the v is nonterminal **then**
            Choose at random an action a from v that is compatible with s and leads to a node v' not on the transposition table
            Add v' as a child node to v
        end if

        // Simulation
        Run a simulation compatible with d from v' until a terminal state

        // Backpropagation
        for each node visited during this iteration **do**
            Update the node's visit count and total simulation reward Update the availability count of the node and its available siblings.
        end for
    end for
    return an action a at root v_0 with the maximum reward
end function

.

5 INFERENCE AGENT

This chapter describes an inference agent designed for the game of Hearts that observes opponent actions to predict their hands and improve decision-making. It builds on the ISMCTS-T agent introduced in Chapter 4, replacing its random sampling method with a more accurate inference-based approach. We will refer to it as the ISMCTS with Transpositions and Inference (ISMCTS-TI) agent.

Hearts is a game with a considerably high disambiguation factor: the number of possible states in a information set quickly shrinks as the round progresses. Near the end of a round, only one possible state remains, effectively making the game one of perfect information. This happens as cards are played and fewer remain, reducing the number of possible states. The cards played can also provide indirect information about the hidden cards. Consider an example where a player does not follow the lead suit: if the lead was a Spades card, we can be certain that the player no longer has any Spades. This, however, involves only a basic level of inference: it simply means the player understands game rules and will not consider illegal states. Proper inference means that a player is capable of predicting the likelihood of a state by considering how often the observed actions would be played in it.

Inference methods yield better results as more actions have been observed, allowing more confident predictions. However, in Hearts the usefulness of inference diminishes as the round progresses. As more information is revealed, the less inference of the remaining hidden information is needed. This occurs because hidden variables become easier to guess as fewer valid possibilities remain. This slowly transitions the game into a perfect information game. Additionally, after a late state is reached, players have fewer actions and may no longer improve their outcome. Therefore, a good inference method has to quickly improve its accuracy to be useful during the earlier and more decisive tricks.

The inference method developed uses the opponent's action history during a round to predict features of their hand. These estimates are calculated by analyzing a large dataset of past game data we collected offline. We can then use these hand estimates to make the probability of a determinization being sampled proportional to its estimated likelihood.

5.1 Generalized state features

This makes collecting accurate data for each state effectively impossible. Therefore, states are generalized to a small set of defining features. In a similar way to (Buro et al., 2009), this work's inference method makes the likelihood of a state being sampled proportional to the likelihood of its features given the actions observed. This way, even when analyzing a state never seen before, we can get a decent estimate of its likelihood.

For every state s, we generalize it to exactly 4 features: what are the special cards of Spades (Q♠ , K♠ and A♠) each player has. $f_i(s)$ is the feature describing what special cards of Spades player i has at state s. The value of $f_i(s)$ has to be one of the following:

1. The player has no special cards of Spades.
2. The player only has the Q♠, but not the K♠ and A♠.
3. The player only has the K♠, but not the Q♠ and A♠.
4. The player only has the A♠, but not the Q♠ and K♠.
5. The player only has the Q♠ and K♠, but not the A♠.
6. The player only has the Q♠ and A♠, but not the K♠.
7. The player only has the K♠ and A♠, but not the Q♠.
8. The player has all the special cards of Spades.

A state s where the Q♠ and K♠ are in player 1's hand, and the A♠ is in player 4's hand would be generalized to:

- $f_1(s)$ = The player only has the Q♠ and K♠, but not the A♠.
- $f_2(s)$ = The player has no special cards of Spades.
- $f_3(s)$ = The player has no special cards of Spades.
- $f_4(s)$ = The player only has the A♠, but not the Q♠ and K♠.

Other cards were not considered because these three impact the round's outcome the most. The Q♠ is worth half of the round's points, so determining who might have it is essential for avoiding it (or targeting it if you want to shoot the moon). The special significance of the K♠ and A♠ is that they are the only two cards that win against the Q♠. This makes them dangerous to hold, as the player might be forced to play them and end up taking the Q♠. Their significance diminishes to that of any Spades card once the Q♠ is out of play. Therefore, player actions done after the Q♠ exits the game are not used for their inference. These cards considerably determine their respective player's behavior

during a round, this facilitates inference of who might hold them.

5.2 Action history abstraction

Now that the features are described, we need a way to calculate $P(f_i(s)|h)$, the likelihood of feature $f_i(s)$ given an observed action history h. For each player i, their opponents' actions provide little insight into player i's hand, since it is hidden from the opponents. Thus, to predict player i's most likely hand features given h, we analyze only player i's action history, denoted as h_i.

The possible action histories form a very large set, so each action history is abstracted. This discards less relevant details while preserving critical actions that provide significant insight into a player's strategy. These actions are critical because they represent decisions that would be optimal in only a few scenarios, and they strongly determine the set of state features that are more likely. These key elements of player i's action history are captured by the following properties:

1. Number of times i could've played the Q♠ without the risk of taking it back but didn't.
2. Number of times the player could've played the K♠ without the risk of taking the Q♠ but didn't.
3. Number of times i could've played the A♠ without the risk of taking the Q♠ but didn't.
4. Number of times i played the K♠ or A♠ when there was no risk of taking the Q♠.
5. If i played the K♠ when there was a risk of taking the Q♠.
6. If i played the A♠ when there was a risk of taking the Q♠.
7. If i had spare Spades card after taking a risk.
8. Number of tricks i lead.
9. Number of tricks i lead with a card of Spades that wasn't special.
10. Current trick number.
11. If the K♠ is still in the game.
12. If the A♠ is still in the game.

Properties 10, 11 and 12 are global features of a round. They are stored together with player specific properties in order to capture in what round context a player i's action

history was reached when looking back at it. These properties are designed to capture general patterns in player i's actions. How much risk and caution each player displayed serves as a strong indication of their cards. A player who acted cautiously to avoid taking the Q♠ likely does not have it. Conversely, a player who took riskier actions likely holds the Q♠ and thus does not fear other players as much. If a player wasted good opportunities to play a special Spades card X, they are likely not to have X. If a player started a trick with a low-rank Spades card, they are likely trying to setup a scenario where the Q♠ would be played and be given to someone else, so they probably don't have it.

5.3 Probabilistic inference

To collect data for feature likelihood estimation, 9,000 games were played offline between 4 ISMCTS-T agents from Chapter 4 with 16k iterations per search. The number of times an agent reached an action history h_i with a hand defined by feature f_i was stored. Hence, to estimate $P(f_i(s)|h_i)$, the method looks up the ratio of times h_i was reached by a player with a hand defined by $f_i(s)$ to the total times h_i was reached.

Now that there is a way to map action histories to state feature likelihoods, Equation 5.1 is used to estimate the likelihood of any set of state features. It is very similar to Equation 2.4 in that it combines probabilities of individual features to calculate the overall likelihood of a state.

$$P(f \mid h) = \prod_{i=1}^{4} P(f_i \mid h_i) \tag{5.1}$$

However, some sets of state features, despite being illegal, are still calculated to have non-zero likelihood. For example, a set where players 1 and 3 have the Q♠, or a set where player 1 has the Q♠ even though it is known not to have any Spades. The likelihood of these illegal sets will be assigned 0. When this occurs, the total likelihood of all legal sets will not sum to 1.0. Therefore, the result must be normalized by dividing each set's likelihood by the total likelihood of all legal sets.

After calculating a final estimate of $P(f \mid h)$, the percentage of states sampled with the set of features f will be proportional to it.

5.4 Exploitability and robustness

To make the inference agent more robust, three sampling mechanisms were added:

- 95% of all determinizations are sampled using the inference method Section 5.3. The remaining 5% continue to be random.
- For every legal set of state features, it is guaranteed to sample at least 1 determinization with it.
- If the current action history has been reached by the legal sets of state features less than 10 times, an older action history containing less recent information, but that has been reached 10 times or more, is used instead.

6 EXPERIMENTAL RESULTS

In this chapter, the ISMCTS-T and ISMCTS-TI agents proposed in Chapters 4 and 5 are evaluated by how well they achieved their particular aims and performed in the game. Section 6.1 describes the testing environment: what constants were used, how hand strength was balanced between the rounds run locally and how performance was measured. Lastly, it introduces the website where ISMCTS-TI faced human opponents. Section 6.2 presents how the use of transpositions impacted search space exploration and performance. Section 6.3 shows how many opportunities there were for the ISMCTS-TI agent to improve decision-making, how good its accuracy was, how its better state sampling enhanced performance, how robust it was to unseen behavior and how it performed against human players.

6.1 Testing environment

The project was entirely implemented in C++. The experiments were conducted on a laptop equipped with an Intel Core i7-8565U CPU (1.80GHz, 1.99GHz boost), 8GB of RAM, and a 64-bit Windows operating system. The experimental setup uses c=0.9 (exploration constant) and k=1,000 (number of determinizations to be sampled for use during search) for all algorithms. When appropriated, reported confidence intervals are 95%.

6.1.1 Local testing environment

While locally testing the algorithms designed, two simplifications of the game were made.

- **No passing phase:** Passing cards was not the focus of this work and would make experiments take longer to run. Therefore, tests were run on the 4th round of the game cycle, when players keep their cards.
- **Game threshold and player order not taken into account:** Normally, a game ends when a player accumulates more points than a certain threshold, then the player with the fewest points wins the game. This adds certain intricacies the players have to consider. If the player is winning, it will want to give points to the

player with the most points, so that the threshold is crossed and it is declared the winner. If it is losing, it will try to delay the game long enough for it to be able to turn the tables around. While expressing these different goals would be possible with a reward function, that would add another degree of complexity to tests. Besides, running multiple rounds until there is a winner can take between 4 and 31 rounds, and it would take much longer to shrink the confidence interval. Instead, all the agents tested only concern themselves with achieving a generally good performance during test rounds.

The performance of each of the algorithms tested was measured by its average reward after a long series of rounds. We also collected what percentage of the points they took and the number of times each agent shot the moon. Every 4 rounds, the cards are shuffled and dealt into four hands. These hands are then distributed to the players. After the round ends, the hands are rotated. Specifically, each player will use the hand that was previously played by the player to their left. This process continues in a cycle, which ensures that each player's performance is not influenced by hand-specific luck.

6.1.2 External testing environment

Test games against human players were conducted on PlayOk[1], a website where many classic board and card games can be played against live opponents in real-time. It gives players a ranking based on their past performance, in a manner similar to the ELO system [2] used in Chess. As Hearts is a multiplayer game, only the game's winner's ranking rises, while the other three players' ranking falls. An exception occurs if a player's time runs out or it forfeits the match, in which case the remaining three players are declared the winners. We found no formal description of the ranking system. Additionally, we found no terms explicitly prohibiting AI testing on the platform.

Only the ISMCTS-TI agent was tested against humans. It needed to be adapted for the full version of Hearts played online, which included the passing phase. Therefore, a simple heuristic agent was developed to pass cards. For every set of cards it could pass, the agent evaluates all possible sets of cards it could receive in return. It then stores the 1,000 most damaging outcomes, as determined by a heuristic function. The set of cards

[1] Link to play Hearts in the PlayOk website <https://www.playok.com/en/hearts/#>.

[2] Wikipedia article explaining the ELO rating system <https://en.wikipedia.org/wiki/Elo_rating_system>.

to pass with the least damaging outcomes is selected. This passing strategy is the AI's weakest component, as it is less refined than the algorithms developed for the playing phase. It also selects cards too cautiously, making shooting the moon almost impossible.

A human operator conveyed the dealt cards and opponent actions to the AI agent, and the AI's decisions were then executed on the website by the operator. This communication caused significant delays, leading the AI agent to frequently be low on time, even though it could make decisions quickly. Thus, all games had to be played with an extended time limit (15 minutes) and a low score threshold (75 points) so the game would end before the agent could run out of time. These constraints made the testing process lengthy and placed the AI agent at a disadvantage, as it had significantly less time to make decisions compared to its human opponents. As a result, each test lasted an average of 18.65 minutes, occasionally exceeding 30 minutes. The constant attention required from a human operator further limited the number of test games conducted.

6.2 Experiments with ISMCTS-T

To experiment, ISMCTS-T played against vanilla ISMCTS opponents (Algorithm 1), which does not account for transpositions. The experiments and results relating to search space exploration are presented in Subsection 6.2.1. To measure performance, 4,000 rounds were run with each possible configuration of 1 to 3 ISMCTS-T agents on the table. The results are detailed in Subsection 6.2.2.

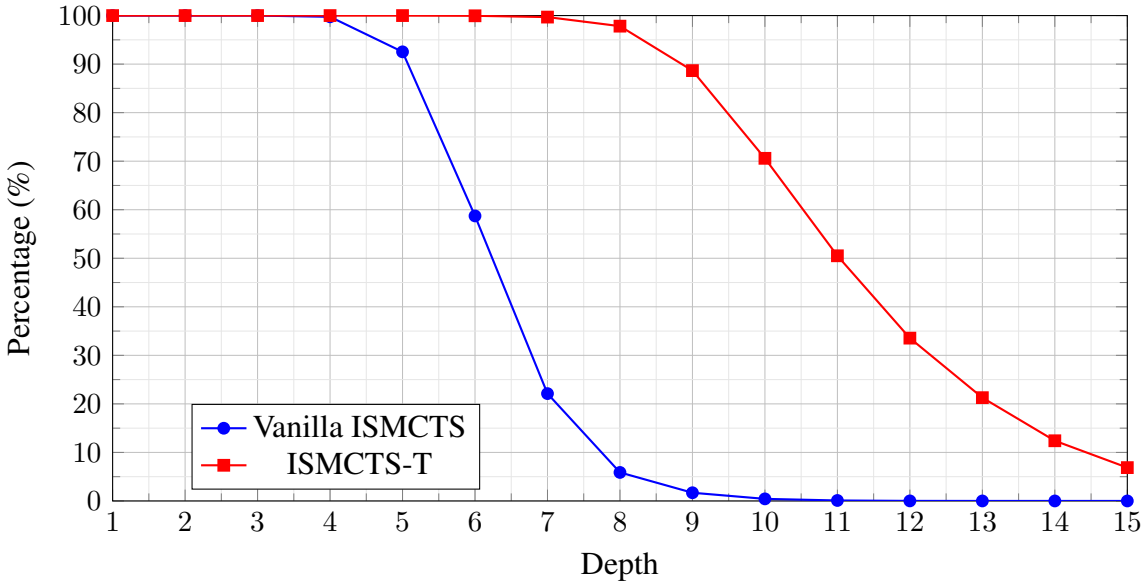
6.2.1 Impact on search space exploration

When we refer to a depth d, it means the searching player explored d actions ahead of the root. Therefore, for a new trick to be fully explored, d has to grow 4 plies. We avoid collecting data from searches that reach terminal nodes too early, since having few remaining actions in a round limits how much further ahead each search can look. This would skew the results, making the algorithms' potential search depth appear shallower than they actually are. Consequently, we only collected data from searches whose root was reached before the 8th trick.

In the first test, an ISMCTS-T agent and three vanilla ISMCTS agents played 4,000 rounds with a budget of 8,000 iterations and no time limit. Figure 6.1 shows the

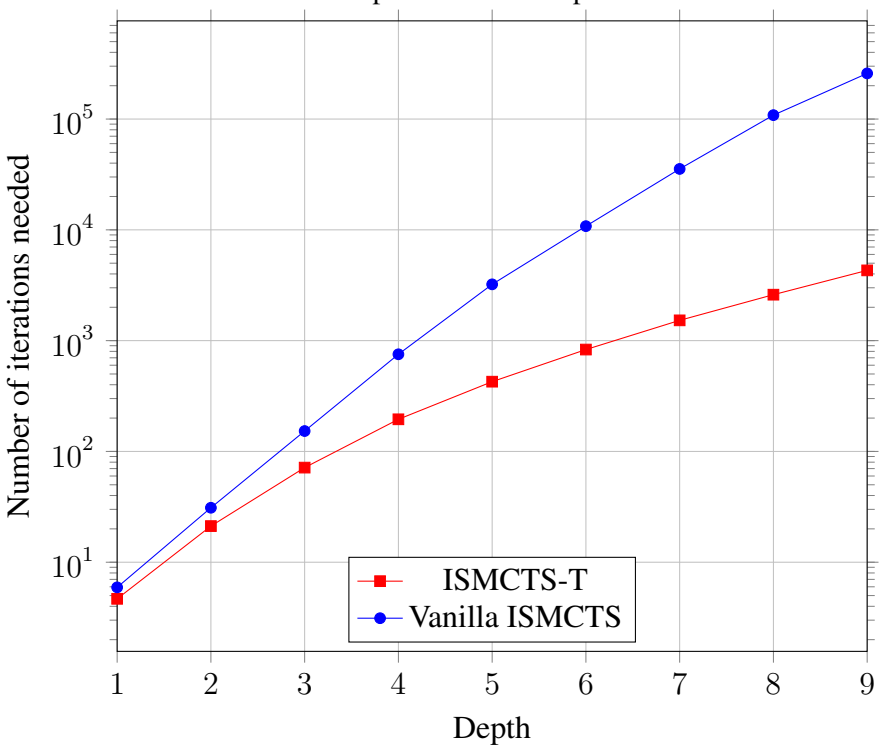
percentage of each agent's searches that explored to a certain depth. After 70k searches, the agents that ignore transpositions only explored a depth of 13 once and never explored further than it. The agent that recognizes transpositions consistently managed to explore way further, even surpassing a depth of 15 in 6.8% of all of its searches.

Figure 6.1 – Percentages of searches that explored to a certain depth with a budget of 8,000 iterations. Results were collected from 4,000 rounds. ISMCTS-T demonstrated superior efficiency, achieving search depths nearly twice as deep as those reached by vanilla ISMCTS.



In the second test, two ISMCTS-T agents and two ISMCTS agents played 100 rounds with no iteration budget and time constraints. Searches continued until a depth of 9 was reached. Figure 6.2 illustrates the average number of iterations required by each agent to achieve specific depths during the first seven tricks. ISMCTS-T demonstrated superior computational efficiency, requiring far fewer iterations than ISMCTS at every depth. For instance, to reach depth 9, ISMCTS required approximately 258,000 iterations, while ISMCTS-T needed only about 4,300, a reduction of over 98%.

Figure 6.2 – Average number of iterations needed to explore to a certain depth. ISMCTS-T demonstrated a much greater efficiency than Vanilla ISMCTS, needing just 1.66% of the budget of its counterpart to reach a depth of 9.



6.2.2 Performance Analysis of ISMCTS-T against ISMCTS

In this subsection, we analyze how well the ISMCTS-T agent performs against vanilla ISMCTS agents when there are different numbers of opponents that are identical to it.

6.2.2.1 A single ISMCTS-T agent against 3 ISMCTS agents

Results are shown on Table 6.1. ISMCTS-T ended with an average reward of 1.27 ± 0.92, these results strongly suggest that ISMCTS-T outperforms its counterpart that ignores transpositions. It shot the moon 105 times with an average frequency of once every 38.09 rounds.

6.2.2.2 Two ISMCTS-T agents against two ISMCTS agents

There are multiple ways to position the two ISMCTS-T agents relative to each other on a table: They can either be side by side (which happens 2/3 of the time), or facing each other (which happens 1/3 of the time). Therefore, instead of showing results

Table 6.1 – Results of one ISMCTS-T agent against 3 ISMCTS agents after 4,000 simulated rounds

Agent	*Table position*	*Average Reward*	*Moon Shots*
ISMCTS-T	1	1.27 ± 0.92	105
ISMCTS	2	0.15 ± 0.94	78
ISMCTS	3	-0.47 ± 0.96	103
ISMCTS	4	-0.95 ± 0.93	103

Source: Author

for a single positioning of agents on a table, results will be aggregated for each type of agent. The ISMCTS agents got an average reward of -0.88 ± 0.68, and shot the moon 200 times (100 each). The ISMCTS-T agents got an average reward of 0.88 ± 0.66, and shot the moon 216 times (108 each). Based on these results, we can confidently claim that the two ISMCTS-T agents outperformed the two ISMCTS agents.

6.2.2.3 Three ISMCTS-T agents against a single ISMCTS agent

Results are shown on Table 6.2. The ISMCTS agent ended with an average reward of -1.32 ± 0.96. These results strongly suggest that it performed worse than the three ISMCTS-T agents it was playing with.

Table 6.2 – Results of three ISMCTS-T agents against a single ISMCTS agent after 4,000 simulated rounds

Agent	*Table position*	*Average Reward*	*Moon Shots*
ISMCTS-T	1	0.47 ± 0.92	89
ISMCTS-T	2	0.82 ± 0.93	91
ISMCTS-T	3	0.03 ± 0.94	89
ISMCTS	4	-1.32 ± 0.96	88

Source: Author

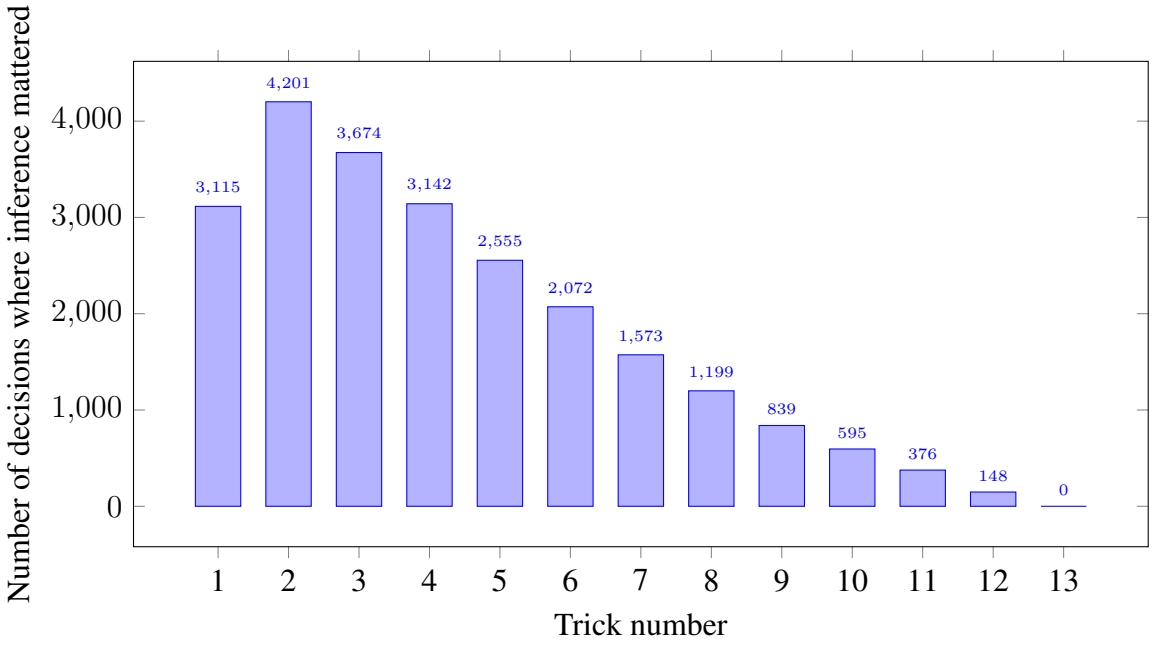
6.3 Experiments with ISMCTS-TI

In this section, we first analyze the accuracy of the predictions made by the ISMCTS-TI agent. We then report on its performance against ISMCTS-T agents, test its robustness against previously unseen behavior, and finally, present its performance against human players.

6.3.1 Sampling accuracy

In these experiments, an ISMCTS-TI agent 5 played 9,000 matches against 3 ISMCTS-T opponents, which are incapable of inference. All the measurements of sampling accuracy are collected from decision points where inference mattered. Inference does not matter if the player only has 1 available action to choose from and/or if it already know the correct state. Figure 6.3 shows the number of times inference mattered to the agent across those games. We can see the number of occurrences drop as the round progresses, this happens because as information is revealed through the round, there are fewer decision points when the player is uncertain and has multiple options. The first trick does not follow this trend even though it is the trick when the player is most uncertain, because 1/4 of the time the player will be the one who is forced to play the 2 ♣. The last trick (13th) never matters, as by then the player has no choice but to play its last card, so it will be excluded on the following plots. The agent collected 2,000 samples at each decision point, 1,000 at random and 1,000 with the help of inference; this way, we can compare how accurate each sampling method was after facing the same scenarios.

Figure 6.3 – Number of decision points where inference mattered by trick number. Inference mattered only when the player had multiple actions available and was uncertain about the correct state. The data was collected from an ISMCTS-TI agent over 9,000 rounds.

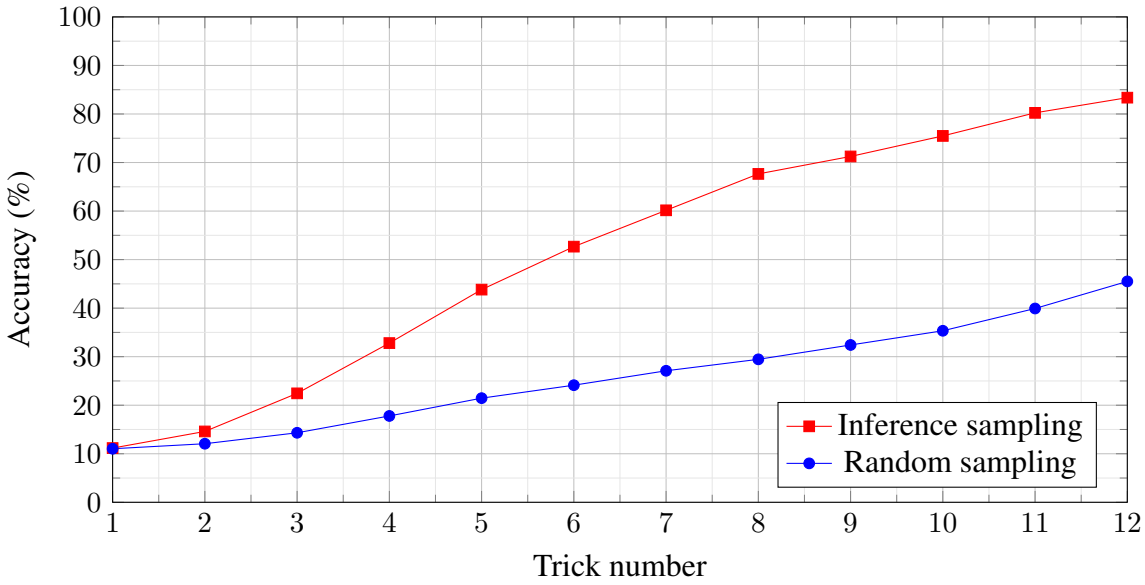


We define the accuracy of a determinization sampling method by the percentage of states sampled with it that matched the features (defined in section 5.1) of the true state. Other states with the slightest mismatch are considered incorrect samples. However, their

impact is likely positive due to their resemblance to the correct state. We must remember that of all states sampled by the inference method, 5% are random, so it can never surpass 95% accuracy.

Figure 6.4 shows the average accuracy of the random and inference sampling methods across tricks. At the start, they both share the same accuracy, as few actions have been played for our inference method to analyze. As the game progresses, their accuracy increases as there are fewer and fewer possible legal states to consider. However, our inference sampling method is shown to improve on top of that, becoming around 2x more accurate than the random sampling method.

Figure 6.4 – Average prediction accuracy by trick number. Inference sampling becomes approximately twice as accurate as random sampling as the game progresses.



However, even though our inference method has a higher accuracy on average, it won't always make the correct inference. Figure 6.5 shows the percentage of times when the inference method gave a higher than average likelihood to the actual state features. It is shown to be an improvement in the majority of times, as even during the first trick, it provides an estimate that tends to be slightly more accurate then average.

As shown by Figure 6.3, most decision points where inference matters happen in the first few tricks. Thus, the high accuracy our method achieves on the latter tricks is not as impactful on performance. Figure 6.6 demonstrates this, aggregating data from all 9,000 rounds. The green line shows the total number of states sampled (1,000 per decision). The red line shows the total number of correct samples by the inference method. The blue line shows the total number of correct samples by the random sampling method. As we can observe, the great majority of decisions happen while our accuracy is still low.

Figure 6.5 – Percentage of times when our inference method gave a higher than average likelihood to the correct state by trick number. Inference is shown to be an improvement in the majority of cases when it is applied.

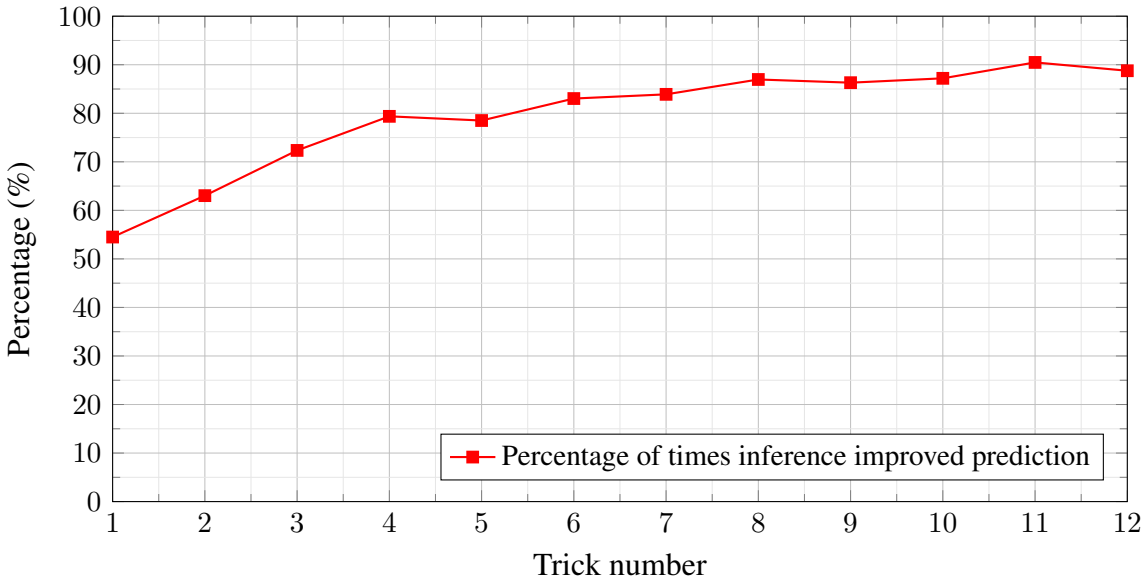
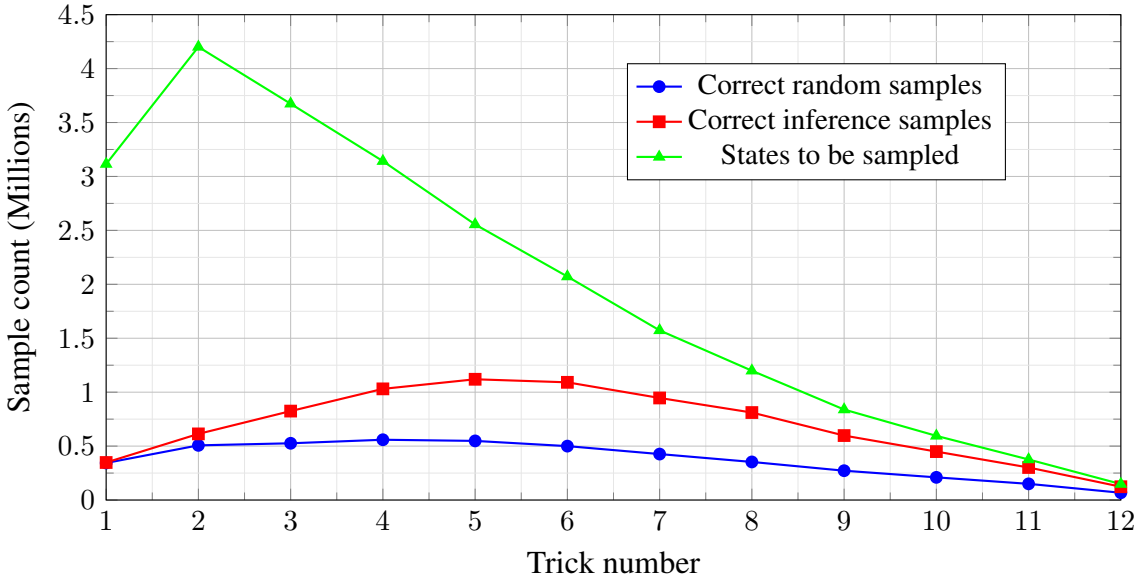


Figure 6.6 – Number of states to be sampled and the number of correct samples by the inference and random sampling methods, broken down by trick number. The green line represents how many states had to be sampled (1,000 per decision), ideally, the number of correct samples should be close it. The chart highlights that most decisions requiring samples occur early in the round, when the accuracy of the inference method is still relatively low. The data was collected from an ISMCTS-TI agent over 9,000 rounds.



From a total of 23,489,000 states sampled, the inference method sampled 8,256,824 of those states correctly (35%), while random sampling sampled the correct state only 4,462,770 times (19%). The total number of states sampled acts here as the upper bound for inference, it seems unlikely that any inference method could go much higher given the very limited information available during earlier tricks.

6.3.2 Performance analysis of ISMCTS-TI against ISMCTS-T

In our performance experiments, a single ISMCTS-TI played against 3 ISMCTS-T opponents for a total of 10,000 rounds. As the offline data used for inference was collected from the same opponent agents that were faced, this inference method probably will not work as well against other types of opponents whose playstyles it has not seen. Nonetheless, most human players play Hearts following the similar overall strategy, as the game involves little inference and bluffing, so the results can likely be generalizable.

ISMCTS-TI got an average reward of 0.46 ± 0.60. The inference agent demonstrates potential for positive performance, but the wide confidence interval indicates that 10,000 test rounds were insufficient data to conclusively determine its effectiveness. Further testing by playing more rounds is needed to reduce uncertainty and confirm the agent's performance.

It shot the moon 280 times (on average once every 35.71 rounds), which is more than its 3 opponents who shot it: 271, 249 and 269 times, respectively. It appears that being able to accurately predict the 3 highest cards of Spades made it easier for it to shoot the moon, but again, moon shots are a rare event to draw strong conclusions after just 10,000 rounds.

The positive impact of our method of inference would be lesser if the passing phase was not skipped. By passing 3 cards to an opponent, the player is not only improving their hand by discarding cards that do not mix well, but also retaining the knowledge of who has those cards. If the Q♠, K♠ or A♠ are passed, inferring where they are would not be needed, as the player can be sure of who has them, so ISMCTS-TI would perform closer to, or the same as, ISMCTS-T.

6.3.3 Robustness

To test the robustness of the ISMCTS-TI agent, we set it and a ISMCTS-T with a budget of 8,000 iterations against ISMCTS-T opponents with a smaller budget of 50 iterations. The opponents were given a small budget to encourage semi-random behavior: the predictable actions are more likely to be played, but the Monte Carlo nature of ISMCTS means that even sub-optimal actions are occasionally selected. This matchup was repeated for 4,000 rounds for both the ISMCTS-T and ISMCTS-TI agents, in order to test how this unseen and unpredictable behavior negatively impacts the inference agent's performance in comparison to ISMCTS-T.

The ISMCTS-T agent achieved an average reward of 9.27 ± 0.77. It shot the moon just 47 times, while its opponents shot it 58, 60 and 63 times, respectively.

The ISMCTS-TI agent achieved an average reward of 9.09 ± 0.75. It shot the moon just 32 times, while its opponents shot it 46, 52 and 56 times, respectively.

Both approaches performed nearly identically. The unseen behavior caused the inference agent to lose the edge it previously displayed against the no-inference agent. However, it was not misled to the point of performing significantly worse. Thus, it can be concluded that the inference agent demonstrates some level of robustness against new behavior.

The opponents likely shot the moon more often due to poor play. Since they were less effective at avoiding taking points, they had more opportunities where they could try to turn a significant loss into a moon shot. They also attempted it more frequently in scenarios where it was likely to fail, as they lacked sufficient simulations to correctly evaluate its viability.

6.3.4 Performance analysis of ISMCTS-TI against human players

To test how ISMCTS-TI performs against human players, 20 games were played in the PlayOk website. While the limited number of games does not allow for statistically significant conclusions, as luck plays a large factor in Hearts, these results provide preliminary insights into the algorithm's performance. Out of the 20, the agent won 9, placed second on 5, placed third on 2 and placed forth (last) on 4. Achieving a 45% win rate, higher than that of most humans on the website. However, 7 of its wins were due to one of its opponents abandoning the game; these wins cannot be disregarded, as most opponents

who abandoned the game did so after they were losing to our agent. Its ranking dropped from 1200 to 1186, likely a consequence of the many game abandonments by opponents being less weighted by the ranking system than a victory.

Throughout all 20 games, the agent did shoot the moon, mostly likely due to the passing agent used (described in Subsection 6.1.2) having a very conservative playstyle. Despite this, the agent's overall performance was not significantly affected. When an opponent attempted to shoot the moon, the ISMCTS-TI agent rarely attempted stopping it. This is likely a consequence of an opponent shooting the moon not being the worst case scenario according to its reward function (see table 4.1). Additionally, it is probable that the algorithm's sampling underestimated the hand strength of players attempting this strategy, as randomly sampled states rarely support shooting the moon. Interestingly, this behavior often resulted in other opponents assuming the responsibility of stopping the moon shot, sparing the ISMCTS-TI agent from the need to make such sacrifices.

Figure 6.7 illustrates the stats of the human opponents ISMCTS-TI played against, and how well it did against them. When a player is said to outperform another, it means their final placement once the game concluded was better. The agent was considered to have drawn/tied against an opponent when they share a victory after another player runs out of time or abandons the game. It can be seen that our agent was able to outperform some experienced humans players that had played more than 10,000 matches. Additionally, it was rarely outperformed by players with low rankings. However, it was either outperformed by or tied with all players with a ranking above 1343. Table 6.3 aggregates the results by opponent performance against the ISMCTS-TI agent. While the agent outperformed 18 opponents and was outperformed by 21, the 7 opponents who abandoned the game were losing to our agent prior to quitting. These findings suggest that the ISMCTS-TI agent demonstrates promising performance, particularly against mid-tier opponents, despite the limited sample size of games.

Figure 6.7 – Scatter plot depicting the stats of opponents faced by ISMCTS-TI in 20 games. The Y-axis represents the ranking of opponents, while the X-axis represents the number of games they have played on PlayOk. Green squares indicate opponents outperformed by the ISMCTS-TI agent (ended the game with a worst placement than it), while red triangles indicate opponents who outperformed it (ended the game with a better placement than it). Blue circles represent opponents who abandoned the game, and cyan diamonds indicate opponents that tied with the ISMCTS-TI agent after another opponent's abandonment.

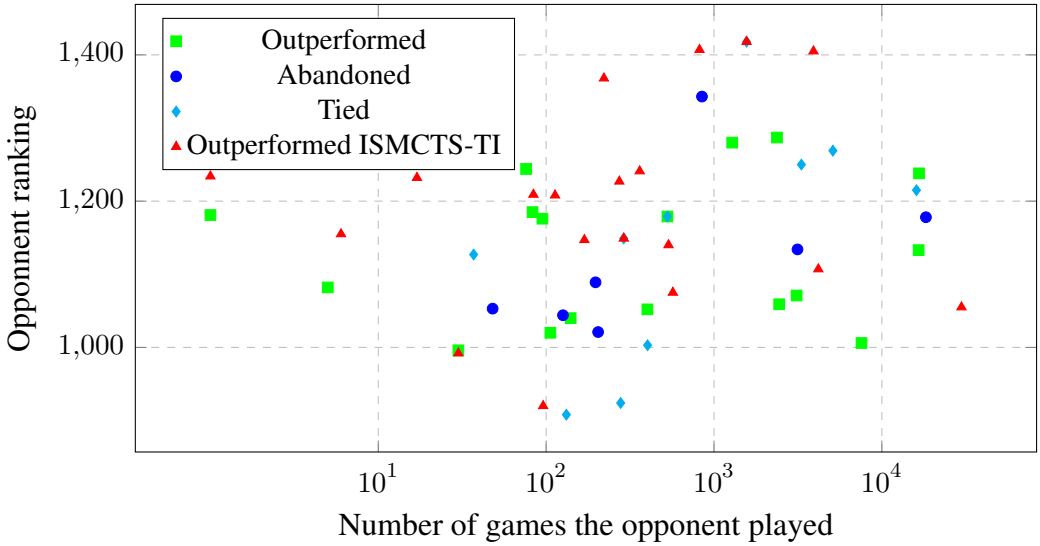


Table 6.3 – Aggregate performance of the ISMCTS-TI agent against human players on PlayOk, categorized by opponent outcomes. The total count, the average opponent ranking and the median number of games played by opponents are shown for each outcome

Result	*Count*	*Average Ranking*	*Median number of games played*
Outperfomed	18	1127.78	464.5
Abandoned	7	1123.14	204
Tied	14	1135	528
Outperformed ISMCTS-TI	21	1185.67	273

Source: Author

6.4 Summary of Results

Both proposed agents showed promising results. The ISMCTS-T agent significantly outperformed the vanilla ISMCTS by exploring the search space more efficiently. This enhanced efficiency led to a noticeable improvement in its overall performance. In contrast, while the ISMCTS-TI agent demonstrated increased accuracy in its state sampling, the observed improvements were not sufficient to conclusively surpass the performance of the ISMCTS-T agent. Therefore, we cannot confidently claim that ISMCTS-TI

provides a clear advantage over ISMCTS-T in its current form.

The ISMCTS-TI agent demonstrated competitive performance against experienced human players, but additional tests are required to obtain a more reliable assessment. It is likely that the ISMCTS-TI agent, which was not developed with the capability to pass cards, was negatively affected by being coupled with a less refined passing agent. It is also very probable that the ISMCTS-TI agent could have turned its many second placements into victories if its reward function attempted to win the game by giving points to the opponent in first place instead of attempting to get a good round per round score.

7 CONCLUSION

We introduced two novel Information Set Monte Carlo Tree Search (ISMCTS) algorithms tailored to the imperfect information game of Hearts. The first, ISMCTS with Transpositions (ISMCTS-T), leverages a transposition mechanism capable of detecting strategically identical information sets. This enhancement aimed to reduce the computational redundancy, improving search efficiency. The second algorithm, ISMCTS with Transpositions and Inference (ISMCTS-TI), incorporates probabilistic inference to estimate the features of the opponents' hands based on their past actions, improving the prediction of hidden information.

Our experiments demonstrated that the transposition detection mechanism significantly improved search space exploration efficiency, as evidenced by its deeper search depth and the superior performance of the ISMCTS-T agent that applies it compared to the vanilla ISMCTS. Although ISMCTS-TI improved state sampling accuracy, its performance did not clearly surpass ISMCTS-T. Notably, ISMCTS-TI demonstrated competitive play against experienced human players. However, its performance may have been hindered by three factors: being coupled with a simple heuristic card-passing agent, using a reward function that prioritized consistent round scores over improving game placement and limited time constraints imposed by the requirement of an intermediary human operator.

Future research could explore several directions. The baseline ISMCTS algorithm has to be modified so it does not assume opponents have access to the agent's private information. The transposition detection system could abstract card suits, potentially improving search space efficiency by another order of magnitude. The inference method could be extended to predict additional state features and to incorporate more available knowledge, such as what cards were passed by the opponent. Enhancing the algorithm used for passing cards is a priority, as while our agents performed well during the playing phase, early suboptimal decisions during the passing phase have likely hindered it when playing the full version of Hearts online. Additionally, revising the reward function to strategically prioritize winning the game, such as targeting the leading player, could also turn many second-place finishes into victories. Expanding the experimental evaluation with a larger sample size and more diverse opponents will also provide a clearer understanding of the algorithm's capabilities and limitations. By addressing these challenges, ISMCTS-TI could achieve significantly higher performance in Hearts and serve as a foun-

dation for advancing AI research in other imperfect-information games, opening the door to broader applications in competitive AI.

REFERENCES

AUER, P. Using confidence bounds for exploitation-exploration trade-offs. **Journal of Machine Learning Research**, v. 3, n. Nov, p. 397–422, 2002.

BAX, F. **Determinization with Monte Carlo Tree Search for the card game Hearts**. Dissertation (B.S. thesis), 2020.

BROWN, N.; SANDHOLM, T.; MACHINE, S. Libratus: The superhuman ai for no-limit poker. In: **IJCAI**. [S.l.: s.n.], 2017. p. 5226–5228.

BROWNE, C. B. et al. A survey of monte carlo tree search methods. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 4, n. 1, p. 1–43, 2012. ISSN 1943-068X.

BURO, M. et al. Improving state evaluation, inference, and search in trick-based card games. In: **Twenty-First International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2009.

COWLING, P. I.; POWLEY, E. J.; WHITEHOUSE, D. Information set monte carlo tree search. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE, v. 4, n. 2, p. 120–143, 2012.

COWLING, P. I.; WHITEHOUSE, D.; POWLEY, E. J. Emergent bluffing and inference with monte carlo tree search. In: IEEE. **2015 IEEE conference on computational intelligence and games (CIG)**. [S.l.], 2015. p. 114–121.

DEMİRDÖVER, B. K.; BAYKAL, Ö.; ALPASLAN, F. Learning to play an imperfect information card game using reinforcement learning. **Turkish Journal of Electrical Engineering and Computer Sciences**, v. 30, n. 6, p. 2303–2318, 2022.

FINNSSON, H. **Cadia-player: A general game playing agent**. Thesis (PhD), 2007.

FRANK, I.; BASIN, D. Search in games with incomplete information: A case study using bridge card play. **Artificial Intelligence**, Elsevier, v. 100, n. 1-2, p. 87–123, 1998.

GINSBERG, M. L. Gib: Imperfect information in a computationally challenging game. **Journal of Artificial Intelligence Research**, v. 14, p. 303–358, 2001.

KOCSIS, L.; SZEPESVÁRI, C. Bandit based monte-carlo planning. In: SPRINGER. **European conference on machine learning**. [S.l.], 2006. p. 282–293.

LONG, J. et al. Understanding the success of perfect information monte carlo sampling in game tree search. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2010. v. 24, n. 1, p. 134–140.

POWLEY, E. J.; COWLING, P. I.; WHITEHOUSE, D. Information capture and reuse strategies in monte carlo tree search, with applications to games of hidden information. **Artificial Intelligence**, Elsevier, v. 217, p. 92–116, 2014.

REBSTOCK, D. et al. Policy based inference in trick-taking card games. In: IEEE. **2019 IEEE Conference on Games (CoG)**. [S.l.], 2019. p. 1–8.

RICHARDS, M.; AMIR, E. Opponent modeling in scrabble. In: **IJCAI**. [S.l.: s.n.], 2007. p. 1482–1487.

SAFFIDINE, A.; CAZENAVE, T.; MÉHAT, J. Ucd: Upper confidence bound for rooted directed acyclic graphs. **Knowledge-Based Systems**, Elsevier, v. 34, p. 26–33, 2012.

SCHLAGNITWEIT, P. **Inference in card games using the example of Schnapsen**. Thesis (PhD) — University of Applied Sciences Technikum Wien, 2023.

SILVER, D. et al. Mastering the game of go with deep neural networks and tree search. **nature**, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016.

STURTEVANT, N. An analysis of uct in multi-player games. **ICGA Journal**, IOS Press, v. 31, n. 4, p. 195–208, 2008.

STURTEVANT, N. R.; WHITE, A. M. Feature construction for reinforcement learning in hearts. In: SPRINGER. **Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5**. [S.l.], 2007. p. 122–134.

ŚWIECHOWSKI, M. et al. Monte carlo tree search: A review of recent modifications and applications. **Artificial Intelligence Review**, Springer, v. 56, n. 3, p. 2497–2562, 2023.

WHITEHOUSE, D. **Monte Carlo tree search for games with hidden information and uncertainty**. Thesis (PhD) — University of York, 2014.