Analysis of Q-Learning and POMDP Frameworks for Optimal Poker Strategies

Matthew Simpson[1]
[1]*Department of Aeronautics and Astronautics*
Stanford University
mssimpso@stanford.edu

Michael Buchar[2]
[2]*Department of Aeronautics and Astronautics*
Stanford University
mbuchar@stanford.edu

Abstract

This project explores the application of Q-Learning and Partially Observable Markov Decision Process (POMDP) frameworks for developing optimal strategies in poker, which is a game characterized by hidden information and strategic complexity. By leveraging state space abstraction techniques, a rudimentary poker agent was developed using reinforcement learning. The state representation was enhanced to include belief-based modeling of the opponent's hand strength, enabling the agent to make decisions under uncertainty. A baseline Markov Decision Process (MDP) model was compared to a POMDP-based agent that incorporates probabilistic reasoning and belief updates. Additionally, policy optimization methods, including nearest neighbor and kernel smoothing, were explored to improve decision-making in unseen states. The agents were evaluated through self-play and performance metrics, such as win rates. Results demonstrate the advantages of POMDP frameworks in handling hidden information and their potential to improve performance in complex decision-making scenarios.

Introduction

The game of poker involves a bimodal approach to victory, where a rigid understanding of the current strength of a hand is blended with a fluid, informational perception of the strength of an opponent's hands based on their actions during and surrounding gameplay. This poses a challenge for simulation engines, which are unable to fully account for the nuances of opponent actions and instead assume rational gameplay, leading to limitations in defining an optimal strategy.

In contrast, games such as Chess and Go feature a completely observable state space; thus, in theory, brute-force methods can be utilized to determine absolute optimality given unlimited resources. In practice, however, this is impractical due to the immense complexity of these games.

In this project, the benefits of model-free reinforcement learning were explored in the development of a rudimentary poker agent, which was then compared to a more sophisticated POMDP agent to determine the effectiveness of opponent classification within a limited observation space.

Relevant Prior Work

Poker strategy can be separated into two main types for constructing poker agents. The first strategy is the development of a "Game Theory Optimal" agent for the two-player zero-sum game in which policy iteration converges to a Nash equilibrium [6]. At the Nash equilibrium, neither player gains profit as the games play out, but players will not lose money by executing this strategy. This can be thought of as a defensive strategy that does not take into account exploitable characteristics that players may observe from the opponent.

The second strategy involves a more widely researched topic with a larger domain of exploitable strategies, such as Monte Carlo Tree Search [9], which can be computed efficiently due to the limited tree depth of a single poker game [2], and case-based reasoning [10]. These strategies also aim to maximize profit in the game.

A common starting point for the development of a poker agent is the simplification of the state space to extract relevant information that is useful for decision-making. For example, [5] outlines strategies for a matrix representation of the state space given possible actions of the player and opponents for the current and future turns. Similarly, Sandholm [3] utilizes branch-and-bound to determine the relative strength of a hand, and Brown [1] proposes a belief state update using Bayes' Rule to gain knowledge of opponent hand strength.

Modifications to the game rules can also increase the speed at which training data is generated. For example, Sonawane [8] discusses the use of limited deck subsets and a limited action space to prevent re-raising. The concept of "bucketing" states is also discussed by Sonawane [8], which reduces the state size by correlating large state spaces with small action spaces. This approach provides the required information for decision-making without requiring verbose policy mapping.

The most widely known poker agent is the Pluribus agent [2], which utilized self-play, information abstraction [7], and counterfactual regret minimization to determine if a better action than previously taken yielded a higher reward. If this is true, it favors that action in future games. This agent was able to generate an average of $1000 per hour in real gameplay, exceeding the expectations of the researchers.

Based on this background, state space abstraction, along with a model that utilizes a belief representation of the opponent's hand, will be explored in policy generation for Texas Holdem poker. The no-limit action will be discarded based on the limited states in which this action provides strategic benefit.

Problem Definition

To model a game of poker and develop policies, two main objectives must be addressed. The first objective involves abstracting the baseline states of the game to extract valuable information while reducing the total state space, $|\mathcal{S}|$. The second objective focuses on leveraging opponent actions to construct a state-space representation that captures the belief about the opponent's hand, which can then be used for informed decision-making.

A baseline Markov Decision Process (MDP) can be developed without addressing the second objective. However, this project should demonstrate the advantages of using a Partially Observable Markov Decision Process (POMDP) framework to achieve a higher-fidelity state representation for policy development. By evaluating these two models against a baseline and against each other, meaningful conclusions about their performance can be drawn.

Approach

State Space Abstraction

To begin, we assume a completely observable state space that disregards the opponent's hand. The minimum state space $|\mathcal{S}|$ must include the complete set of board cards along with the set of player cards, which are directly observable. This results in the following state space size:

$$|\mathcal{S}| = \prod_{k=46}^{52} k \tag{1}$$

This evaluates to approximately 6.7×10^{11} states, far exceeding what can realistically be trained using reinforcement learning (RL) methods. To reduce the state size, abstractions were made to extract relevant information from the player hole cards, board cards, and their union. Monetary information was also included to inform betting decisions. For a POMDP model, the opponent belief state is incorporated into the state space. The total state space used during modeling is provided in table 1.

The win bucket characterizes the probability of winning a hand using the multi-modal Dirichlet distribution with a 10% prior to ensure that the split pot probability is captured. The prior was generated by simulating 100,000 hands with random hole cards, with each hole card combination simulated for 10,000 hands. The final probability was calculated using the following equation:

$$P_i = \frac{\beta_i}{\sum_{j=1}^{k} \beta_j} \tag{2}$$

where

$$\beta_i = w_i + \alpha_i \tag{3}$$

State	Value
Player Cards	
Win Bucket	(1, 2, 3)
Suited	(0, 1)
Connectedness	(0, 1)
High Card	(0, 1)
Board Cards	
Flop Status	(0, 1, 2, 3)
Paired Board	(-1, 0, 1)
Connected Board	(-1, 0, 1)
Board High Card	(-1, 0, 1, 2)
Count High Cards	(-1, 0, 1, 2, 3, 4, 5)
All Cards	
Rank of Pair	(-1, 0, 1, 2, 3)
Number of Paired Cards	(0, 1, 2, 3)
Flush	(-1, 0, 1, 2)
Straight	(-1, 0, 1, 2)
Full House	(-1, 0, 1)
Monetary	
Pot Size	(0, 1, 2, 3, 4, 5)
Player Bankroll	(0, 1, 2, 3, 4, 5)
Opponent Belief State	
Win Bucket	(1, 2, 3, 4, 5)

Table 1: State and Value Table

Here, w_i represents the observed wins, losses, or splits from the 10,000-game simulation, and α_i is the prior determined wins, losses, or splits, scaled to 10% of the number of simulations used during the win probability calculation.

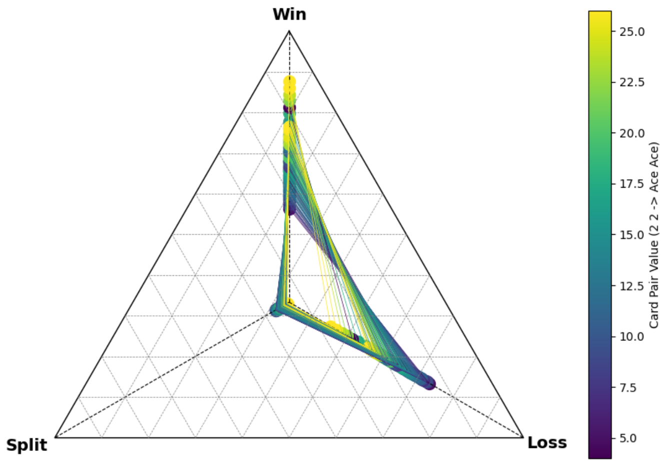

Figure 1: Dirichlet win probability for card rank pairs.

Each state was calculated using individual methods based on the suit and rank of the player and board cards. The pot size and player bankroll were then bucketed, as described by [8], using basic division to inform the model's decision-making based on the stage of the game. The resulting state space contains approximately 7.1×10^7 possible states for the Q-Learning model and 35.5×10^7 states for the POMDP model, representing a 10,000-fold and 2,000-fold decrease in size compared to the original state space respectively. While still large for RL methods, further reductions would significantly impair decision-making capabilities.

Model Definition

State Space The state space for training data simulation was calculated using a single Python method before an action was taken. For the beginning turn known as the pre-flop, states calculated based on the board cards and player cards compared to the board card were set to -1. Depending on the type of agent being generated, specific state spaces must be generated based on the model definitions.

Action Space The player's actions were pseudo-randomized to fit within the bounds of possible actions at each turn of a poker game for the baseline model. Initial actions were determined based on the win bucket and player bankroll, evaluating whether betting was a useful choice. Specific state space possibilities were tailored if a raise or bet was taken, and the raise limit was set to two to prevent multiple re-raises from slowing down training data generation. Betting was bucketed into two categories: a small bet and a big bet. A small bet represented 10% of the bankroll, while a big bet represented 25% of the bankroll. The total action space is:

$$\mathcal{A} = \begin{aligned}&\{\text{Fold, Check, Call, Bet Small,}\\ &\ \text{Bet Big, Raise Small, Raise Big}\}\end{aligned}$$

Transition Model The transition model is broken into two components. The first component handled each turn of play with the action function being called to determine the player and opponent actions. If this is the the pre-flop phase, then minimum bets are placed, and the bankrolls of each player, along with the pot are updated accordingly. A single opponent was selected for simplistic training data generation. Once actions are selected the pot and player bankrolls are updated. If a bet or raise is the selected action, appropriate transition function logic dictates if a player or opponent must select a follow-up action. If no actions are needed, an additional card is added to the board cards to begin the next turn. If this is the final turn, the second component uses a showdown method to determine the winner of the game and update the pot and bankrolls accordingly. If either the player or opponent has no chips left to play, then the game has concluded.

Reward Model The reward model assigns rewards based on hand strength and the action taken during non-showdown turns, as well as the pot size during showdown turns. The POMDP model also used this reward model framework but utilized the opponent's hand strength as another variable in reward generation.

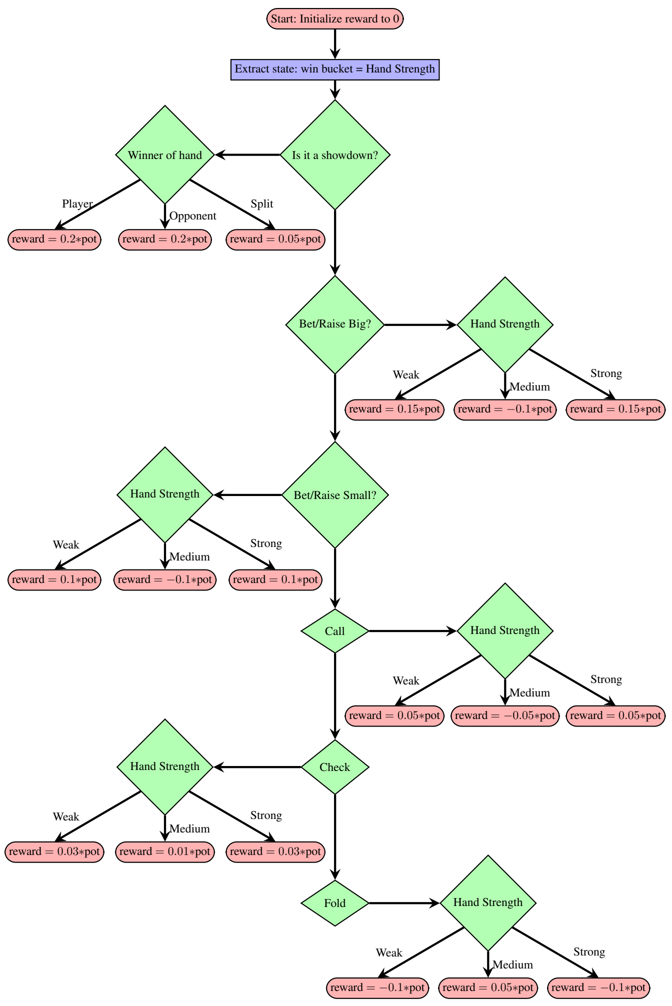

Figure 2: Reward Model for the Q-learning agent.

Reinforcement Learning Training

To generate training data, a logging script was implemented during self-play to log the current state, player action, reward, and next state. The script generated a total of 1,885,285 over an 8-hour period. To generate a baseline policy, Q-learning was implemented based on the function below [4]. The learning rate α was chosen to be 0.1 and the discount factor γ was chosen to be 0.95. 10 Epochs were chosen to revisit the training data and further refine the Q learning model.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \quad (4)$$

A sample output of the Q table has been provided. This table was used to find the optimal action for all training states.

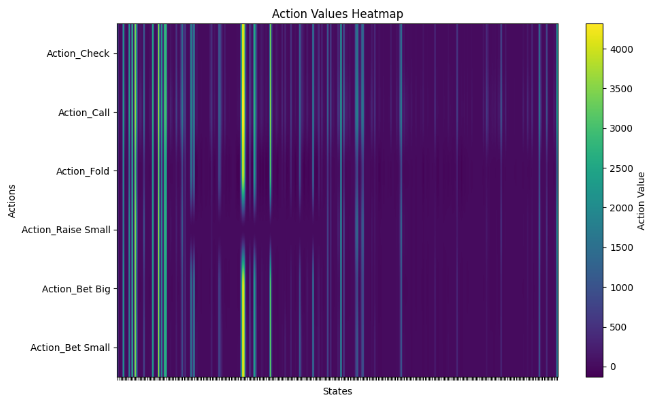

Figure 3: Q Table output from Q-Learning

For states outside of the training data, two policy expansion methods were compared. The first being nearest neighbor found using the following function [4]:

$$a = \arg\min_{j \in \{1,\ldots,m\}} d(s_j, s) \tag{5}$$

where

$$d(s_j, s) = \sqrt{\sum_{i=1}^{n} (s_j - s^2)} \tag{6}$$

In this equation, $d(s_j, s)$ is the distance between the current state and possible states in the state hyperspace with n dimensions. The nearest neighbor can be found by taking the minimum of this equation, and the corresponding action can be found from that nearest state.

The second type of policy expansion explored was Kernel Smoothing. In this approach, the following equation is used to calculate the log density function $\log f(\mathbf{s})$ using the number of states n, the dimensionality of the state space d, and K is the Gaussian kernel function.

$$\log f(\mathbf{s_j}) = \log \left(\frac{1}{nh^d} \sum_{i=1}^{n} K \left(\frac{\|\mathbf{s_j} - \mathbf{s}_i\|}{h} \right) \right) \tag{7}$$

The closest action is found by taking the maximum of this function and finding the corresponding state-action pair associated with that maximum value.

POMDP and Belief Updates

In a Partially Observable Markov Decision Process (POMDP) framework, the agent must make decisions based on an incomplete view of the game state. This is particularly relevant in poker, where the opponent's hand is hidden. To address this, a belief state is maintained and updated as the game progresses, reflecting the agent's probabilistic estimate of the hidden state.

Belief Representation To model the opponent's hand strength, we introduced a bucketed representation with five discrete levels:

- 1: Very Weak
- 2: Weak
- 3: Moderate
- 4: Strong
- 5: Very Strong

These levels are determined based on the opponent's hole cards and the board cards. When the opponent's cards are unknown, their strength is estimated using probabilistic simulations or heuristics. The state representation is updated to include this belief feature, which is dynamically adjusted during the game. This belief serves as a critical component in decision-making under uncertainty, enabling the agent to better anticipate the opponent's possible actions.

Belief Update Algorithm The belief state is updated after each action and observation using the POMDP belief update algorithm. For each possible hidden state s', the belief is updated as a combination of:

- The probability of transitioning to s' from the current state s given the action a, modeled by the transition function $T(s, a, s')$.
- The likelihood of the observation o, given s', modeled by the observation function $O(a, s', o)$.

The updated belief is calculated using the formula:

$$b'(s') = \frac{O(a, s', o) \sum_s T(s, a, s') b(s)}{\sum_{s'} O(a, s', o) \sum_s T(s, a, s') b(s)} \tag{8}$$

After the update, the belief vector is normalized to ensure it sums to 1. This process allows the agent to iteratively refine its estimate of the hidden state based on observed evidence and the rules of the game.

An example updated belief state distribution from the script has been provided for an opponent's actions and an initial belief state of 3 using a basic transition and observation function.

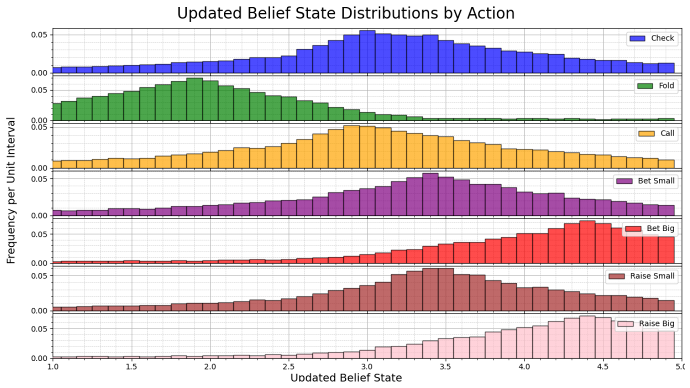

Figure 4: Belief state update distribution across actions for initial b = 3 given we observe the following actions.

Integration with POMDP Framework The belief update mechanism integrates directly with the decision-making process. At each step:

1. The agent observes the game state, including actions taken by the opponent and the resulting game outcome.
2. The belief state is updated to reflect the new information.
3. The updated belief is used to evaluate potential actions and determine the optimal strategy for the current turn.

This iterative adjustment enables the agent to handle uncertainty effectively, leveraging the strength of POMDP frameworks. By combining belief representation, probabilistic reasoning, and strategic decision-making, the agent achieves a higher level of adaptability and performance in complex poker scenarios.

Policy Evaluation

To determine the optimal policy, poker games were played between the initial baseline model with trivial action selection. The two policy expansion methods were also compared.

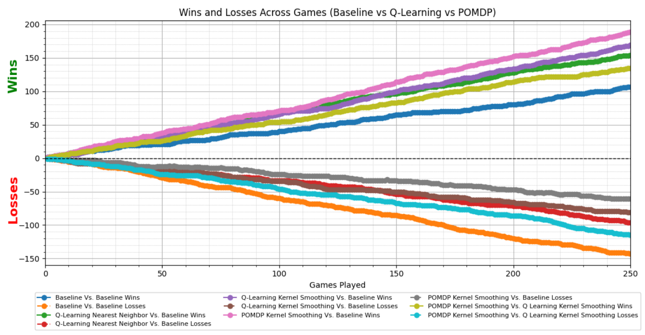

Figure 5: Wins and losses of policies explored

From this evaluation, it can be seen that the Q-learning model outperforms the baseline model using the Kernel Smoothing technique and the nearest neighbor technique, reinforcing the strategy of model training as a viable option for agent training.

Analysis and Results

The performance of the proposed poker agents was evaluated across multiple dimensions, including win rate and the ability to adapt to opponent strategies. This section discusses the results of these evaluations and provides insights into the effectiveness of the Q-Learning and POMDP-based approaches.

Baseline Policy Evaluation

The baseline Markov Decision Process (MDP) model, which relied on trivial action selection strategies, served as a benchmark for evaluating the other agents. This model demonstrated limited success, with a win rate of only **42.8%**. The baseline struggled to adapt to dynamic opponent strategies and performed poorly in scenarios requiring nuanced decision-making, such as betting and bluffing. These limitations underscore the importance of incorporating belief updates and opponent modeling.

Q-Learning Agent Performance

The Q-Learning agent demonstrated moderate success, with a win rate of **67.6%**. The reinforcement learning approach was effective in scenarios with repeatable patterns but struggled in handling hidden information. While the nearest neighbor and kernel smoothing methods improved performance in unseen states, the Q-Learning agent lacked the depth to fully capture the complexities of poker gameplay. Moreover, the Q-Learning agent was less effective against opponents employing dynamic strategies or bluffing. Despite training on over 1.8 million game states, the agent exhibited suboptimal actions in some scenarios.

POMDP Agent Performance

The POMDP-based agent outperformed the baseline model significantly, achieving a win rate of **75.6%**. The POMDP model also outperforms the Q-Learning model with a win rate of **56.0%** using the belief state about the opponent's hand strength. The belief representation of the opponent's hand strength allowed the agent to make more informed decisions under uncertainty. The bucketed hand strength representation enabled the agent to anticipate opponent actions more effectively. Iterative updates to the belief state improved decision-making, especially in scenarios with limited information. Furthermore, the agent demonstrated robustness against both aggressive and defensive opponent strategies. The POMDP agent consistently outperformed the Q-Learning agent in games requiring sophisticated strategy adaptation.

Policy Expansion Methods

Two policy expansion methods—nearest neighbor and kernel smoothing—were compared to address unseen states in the Q-Learning framework. Kernel smoothing performed better, with a **6.0%** higher win rate compared to nearest neighbor. This improvement is attributed to the ability of kernel smoothing to consider the density of similar states rather than relying on a single nearest match.

Key Insights

- **Importance of Belief Modeling:** The inclusion of a belief-based representation of opponent's hand strength proved critical for success, particularly in scenarios with hidden information.
- **Adaptation to Dynamic Strategies:** The POMDP agent's ability to adapt dynamically provided a significant advantage over both the baseline and Q-Learning agents.
- **Trade-offs in Learning Methods:** While Q-Learning was easier to implement and required fewer computational resources, its performance was limited by the lack of probabilistic reasoning and belief updates.

Limitations

Despite the success of the POMDP and Q-Learning agents, several limitations were observed:

- **State Space Size:** Even with state space abstraction, the large number of states posed challenges for efficient training and policy generation.
- **Opponent Assumptions:** The POMDP framework assumed rational gameplay, which may not always reflect real-world opponents' strategies.
- **Limited Exploration of Opponent Types:** The agents were tested against a limited set of opponent types, which may not generalize to all gameplay scenarios.

Conclusion

The baseline MDP agent highlighted the limitations of simplistic strategies in handling dynamic and hidden states, achieving a win rate of only 42.8%. In contrast, the Q-Learning agent exhibited moderate success, leveraging reinforcement learning to reach a 67.6% win rate by exploiting repeatable patterns. However, its limited adaptability to unseen states and opponent strategies underscored the need for a more robust framework.

The evaluation demonstrates the clear advantages of the POMDP framework in handling hidden information and adapting to dynamic strategies with a win rate of 75.6%. While Q-learning showed promise in scenarios with repeatable patterns, its limitations in handling hidden states highlight the need for more advanced approaches. These findings validate the use of POMDP-based models for optimal poker strategy development.

Despite these successes, challenges remain. The large state space, assumptions of rational opponent gameplay, and limited diversity of opponent types in testing highlight areas for future work. Expanding the framework to handle more complex state spaces and incorporating adversarial learning to simulate diverse opponent strategies such as bluffing could enhance performance.

In conclusion, this research demonstrates the potential of combining reinforcement learning with POMDP frameworks to develop sophisticated poker agents capable of navigating the complexities of hidden information and strategic interaction. These findings not only contribute to the field of game AI but also have broader implications for decision-making systems in partially observable environments.

Contributions

This analysis represented a collaborative effort between Matthew Simpson and Michael Buchar, with each contributor playing a key role in the development, analysis and documentation of the poker strategy. Matthew focused on the Q-Learning implementation and the development of policy expansion methods, including nearest neighbor and kernel smoothing techniques. He also set up the majority of the coding framework used to generate training data and evaluate the agents. Michael developed the POMDP framework and worked on belief state modeling and updating, enabling the agent to handle hidden information effectively. The sections of the paper were divided evenly between the two contributors. Matthew and Michael collaborated closely to ensure consistency and coherence throughout the document.

References

[1] Brown, N. 2007. Combining Deep Reinforcement Learning and Search for Imperfect-Information Games. *arXiv preprint arXiv:2007.13544*.

[2] Brown, N.; and Sandholm, T. 2019. Superhuman AI for Multiplayer Poker. *Science*.

[3] Gilpin, A.; and Sandholm, T. 2007. Better Automated Abstraction Techniques for Imperfect Information Games, with Application to Texas Hold'em Poker. *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[4] Kochenderfer, M. J.; Wheeler, T. A.; and Wray, K. H. 2022. Algorithms for Decision Making. *The MIT Press*.

[5] Kuhn, H. W. 1951. A Simplified Two-Person Poker Game. *Contributions to the Theory of Games, Volume I*.

[6] Li, J. 2018. Exploitability and Game Theory Optimal Play in Poker. *Boletín de Matemáticas*.

[7] Sandholm, T. 2015. Abstraction for Solving Large Incomplete-Information Games. *AAAI Conference on Artificial Intelligence*.

[8] Sonawane, P. 2024. A Survey on Game Theory Optimal Poker. *arXiv preprint arXiv:2401.06168*.

[9] Van den Broeck, G.; Driessens, K.; and Ramon, J. 2009. Monte-Carlo Tree Search in Poker using Expected Reward Distributions. *Technical Report*.

[10] Watson, I. 2008. CASPER: A Case-Based Poker-Bot. *International Computer Games Association Journal*.