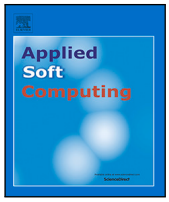
Applied Soft Computing Journal 158 (2024) 111545

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc



Improved learning efficiency of deep Monte-Carlo for complex imperfect-information card games

Qian Luo, Tien-Ping Tan *

School of Computer Sciences, Universiti Sains Malaysia, Penang 11800, Malaysia

ARTICLE INFO

Keywords:
Optimized deep Monte-Carlo
Minimum combination search
Opponent model
DouDiZhu
Big2

ABSTRACT

Deep Reinforcement Learning (DRL) has achieved considerable success in games involving perfect and imperfect information, such as Go, Texas Hold'em, Stratego, and DouDiZhu. Nevertheless, training a state-of-the-art model for complex imperfect-information card games like DouDiZhu and Big2 remains resource and time-intensive. To address this challenge, this paper introduces two innovative methods: the Opponent Model and Optimized Deep Monte-Carlo (ODMC). These methods are designed to improve the training efficiency of Deep Monte-Carlo (DMC) for imperfect-information card games. The Opponent Model predicts hidden information, enhancing the agent's learning speed in DMC compared to the original training that only utilizes observed information as input features. In ODMC, the Minimum Combination Search (MCS) is a heuristic search algorithm based on dynamic programming. It calculates the minimum combination of actions in the current state, and ODMC uses MCS to filter suboptimal actions in each state. This reduces the action space considered by DMC, resulting in faster training that focuses on evaluating the most promising actions. The effectiveness of the proposed approach is evaluated by examining two complex card games with imperfect information: DouDiZhu and Big2. Ablation experiments are conducted to evaluate both the Opponent Model (D+OM and B+OM) and ODMC (D+ODMC and B+ODMC), along with their combined variants (D+OMODMC and B+OMODMC). Furthermore, D+OMODMC and B+OMODMC are compared with state-of-the-art DouDiZhu and Big2 artificial intelligence (AI) programs, respectively. The experimental results demonstrate that the proposed methods achieve comparable performance to the original DMC, but with only 25.5% of the training time on the same device. These findings are valuable for mitigating both the equipment requirements and training time in complex imperfect-information card games.

1. Introduction

Deep reinforcement learning (DRL) [1–3] has demonstrated remarkable performance in a variety of games, including board games, such as AlphaGo [4], AlphaGo Zero [5,6] and DeepNash [7], as well as large video games like AlphaStar [8], OpenAI Five [9], and JueWu [10]. Furthermore, recent studies have achieved notable results in complex, imperfect-information [11] card games like DanZero [12] in GuanDan, DouZero [13] in DouDiZhu, and AlphaHoldem [14] in Texas Hold'em.

The complex, imperfect-information card games are valuable not only because these games attract millions of players daily, but also because they reflect numerous real-world scenarios such as negotiations, surgical operations, business, and physics. Many of these real-world situations can be modeled as games, allowing the methods and strategies refined through the study of these card games to be applied effectively in various practical contexts. However, training a state-of-the-art model for complex, imperfect-information card games such as DouDiZhu and Big2 still requires substantial resources and time due to the following challenges: DouDiZhu and Big2 are imperfect-information games. Agents can only see their own cards and the public cards; they cannot see their opponents' hands. Agents make decisions based on hidden information, which often requires more iterations and time to discover effective strategies. Additionally, DouDiZhu and Big2 have a large and complex state and action space. With the combination of cards and the complex rules, DouDiZhu has approximately 10^{83} potential states and 27,472 possible actions, while Big2 has about 10^{28} states and 19,899 possible legal actions. Consequently, DRL may require more computational resources and time to explore and analyze the multitude of possibilities.

Previous research on complex, imperfect-information card games involves variations of Counterfactual Regret Minimization (CFR) [15], Monte Carlo tree search (MCTS) [16], and Deep Monte-Carlo method (DMC) [13]. CFR adopts a policy based on an action with a minimum

* Corresponding author.
E-mail addresses: steven.luo.stanley@student.usm.my (Q. Luo), tienping@usm.my (T.-P. Tan).

https://doi.org/10.1016/j.asoc.2024.111545
Received 3 September 2023; Received in revised form 14 March 2024; Accepted 19 March 2024
Available online 26 March 2024

regret value in the game tree. However, applying CFR to games with numerous actions and states is impractical [17]. MCTS selects an action as a policy using Upper Confidence Bounds (UCB). UCB is calculated through the repetition of four steps (selection, expansion, simulation, and backpropagation), and this repetition requires a lot of computing resources [18]. DMC combines Monte-Carlo method with deep neural networks and is useful for handling games with large action and state spaces. However, it can be computationally expensive and require a large amount of data to train the model effectively [19]. Therefore, training a high-level artificial intelligence (AI) agent for complex, imperfect-information card games while utilizing reasonable resources and time poses a challenge for small-scale laboratories, individual researchers, and fast deployment.

Contributions: This paper proposes two novel approaches to address the challenges in complex, imperfect-information card games:

1. Opponent Model: An advanced opponent model is proposed to predict imperfect information, which is used as input features along with observable information. Unlike previous opponent model, such as DouZero+ [20], which only predicted the number of cards held by one player, our advanced opponent model goes further. It not only predicts the number of cards in each player's hand but also discerns the minimum composition of these hands, including the suit of each card. To the best of our knowledge, this is the first time such an advanced opponent model has been applied to the Big2 game. Furthermore, compared to the original DMC training, which only utilizes observable information, our agent learns faster with the addition of an advanced opponent model.
2. Optimized Deep Monte-Carlo (ODMC): The Optimized Deep Monte-Carlo (ODMC) is proposed, integrating DMC with Minimum Combination Search (MCS) algorithm. MCS is a heuristic search algorithm based on dynamic programming that calculates the minimum combination of actions in the current state. ODMC employs MCS to filter suboptimal actions in each state. After being processed by MCS, DMC assesses the most promising actions, rather than evaluating all possible actions, resulting in an acceleration of the training process, while maintaining the performance of it.

The proposed approaches undergo evaluation on two complex card games with imperfect information: DouDiZhu and Big2. Through ablation experiments, they assess both the Opponent Model (D+OM and B+OM) and Optimized Deep Monte-Carlo Method (D+ODMC and B+ODMC), as well as their combined variations (D+OMODMC and B+OMODMC). Ablation experiments demonstrate that the proposed methods outperform the original unimproved method on the same server within the same training time. Furthermore, the comparison between D+OMODMC and B+OMODMC against the state-of-the-art benchmarks for DouDiZhu and Big2 AI, respectively, demonstrates that the proposed method effectively trains a professional-level agent to play complex imperfect-information card games in less time.

Outline: Section 2 reviews existing algorithms for imperfect-information card games. Section 3 outlines game environments of DouDiZhu and Big2, and introduces the DouZero AI model. Section 4 details proposed methods, which include innovative Minimum Combination Search (MCS) and integration of Opponent Model with Optimized Deep Monte-Carlo (ODMC) method. Section 5 discusses the experimental setup, conducts ablation studies to assess effectiveness of the proposed methods, and provides a comparative analysis with current state-of-the-art models. Section 6 summarizes the paper and suggests future research directions. Finally, Appendix A offers detailed supplementary information, including specific rules and feature representations for DouDiZhu and Big2, and the MCS algorithm used in DouDiZhu.

2. Related works

Counterfactual Regret Minimization (CFR) is a potent method in game theory and artificial intelligence (AI), iteratively learning from past decision regrets to develop strategies for imperfect-information games, particularly successful in poker, like heads-up limit hold 'em poker [21]. However, implementing CFR to solve complex card games presents significant computational and storage challenges due to the large state and action space of the game. This is because the game tree generated by CFR can become extremely large, especially for games with many possible actions, resulting in high computational costs for traversing the tree. Additionally, storing the regret and average strategy values for each node in the game tree can also require a significant amount of storage space. Monte Carlo Counterfactual Regret Minimization (MCCFR) [18] is a variant of CFR that uses Monte Carlo simulations to randomly sample a subset of the game tree, reducing the computational resources required. However, there may be challenges in using MCCFR to solve complex card games due to its specific rules and gameplay mechanics. For example, DouDiZhu and Big2 have unique rules for ranking hands and playing cards, which may require modifications to the standard MCCFR algorithm. To the best of our knowledge, there have been no successful applications of CFR or MCCFR in complex imperfect-information games such as DouDiZhu and Big2.

Monte Carlo Tree Search (MCTS) is a search-based decision-making algorithm used in AI and game playing that simulates various possible outcomes to iteratively build a tree of actions, helping to make optimal choices in complex decision spaces [16–18]. It calculates the Upper Confidence Bounds (UCB) value for each node to balance exploration and exploitation during decision-making. The UCB value estimates the expected reward of each node, considering both uncertainty and the desire to explore new nodes. By choosing nodes with high UCB values, MCTS strikes a balance between exploring new possibilities and exploiting known information. This enables MCTS to make informed decisions and converge to an optimal solution. However, MCTS faces challenges in games with imperfect information due to unobserved data from other players. Instead, Chen et al. [22] evaluate an approach called Big2AI for Big2 based on Information Set MCTS (ISMCTS) [23]. The algorithm is called "Information Set" because it uses information sets, which are subsets of the game tree that contain all possible outcomes of a player's decision. Similar works have been conducted by Whitehouse et al. [24], who propose a Determinized MCTS for DouDiZhu, by Ihara et al. [25] for Pokémon, by DEMİRDÖVER et al. [26] for Hearts and by Zolboot et al. [27] for Hearthstone. These works share the similarity in requiring state estimation before applying MCTS techniques. However, the unobserved information, such as the hidden cards of the opponents, poses a challenge for accurately estimating game states in MCTS-based approaches. Furthermore, a drawback of MCTS and its variants lies in their high computational cost, particularly in complex games with numerous possible moves.

Opponent model [28] refers to the use of prior information and observations to predict the cards held and future actions of opponents or partners. In extensive-form games with imperfect information, utilizing opponent model to infer hidden information becomes crucial for players to enhance their chances of success. Therefore, the opponent model attracts considerable attention in the field of games. For example, Ganzfried et al. [29] demonstrate the effectiveness of Bayesian opponent model in multiplayer imperfect-information card game Kuhn poker. This approach infers a posterior distribution of opponent strategies and generates an appropriate response based on that distribution. Similarly, Mizukami et al. [30] propose an approach that combines an opponent model and Monte Carlo simulation for another complex imperfect-information game, Mahjong. In this work, the opponent model is trained using expert game records and Monte Carlo simulation. On the other hand, Tang et al. [31] present an approach that combines Evolution Algorithm [32,33] and opponent model for a fighting game. The proposed model, optimized through supervised learning [34–36]

using historical observations from the opponent, achieved success in winning the competition that year. The success of these approaches indicates the potential of opponent model in enhancing the performance of intelligent agents in complex, imperfect-information card games.

Deep Reinforcement Learning (DRL) is an AI approach where agents learn to maximize rewards by making decisions and taking actions in an environment using deep neural networks [1–3]. Unlike CFR and MCTS, DRL can handle complex game environments without prior knowledge or rules. Suphx [37], based on Proximal Policy Optimization (PPO) [38], a popular DRL algorithm, demonstrates superiority over most top human players in Mahjong. The training of each agent in Mahjong, a multi-player imperfect-information game with dozens of possible actions, costs 44 GPUs and takes two days. On the other hand, when applying the PPO method to Big2, it only manages to defeat amateur players [39]. Meanwhile, Guan et al. [40] and Zha et al. [13] find that PPO, Deep Q-Learning (DQN) [41], and Actor–Critic [42] algorithms all struggle to achieve competitive performance in DouDiZhu. Big2 and DouDiZhu are imperfect-information card games with sparse reward, trillions of possible states and tens of thousands of actions. PPO, DQN, and Actor–Critic algorithms are susceptible to overestimation or failure to converge in Big2 and DouDiZhu due to sparse rewards and large action spaces. You et al. [43] attempt to reduce the action space and introduce the Combination Deep Q-Learning (CQL) approach for DouDiZhu. CQL consists of two stages of DQN: the Decomposition Proposal Network (DPN) selects the optimal hand decomposition based on the Q-value, while the Move Proposal Network (MPN) determines the most sensible action from the optimal decomposition. However, despite 20 days of training, CQL fails to outperform simple rule-based heuristics [13]. DeltaDou [44] significantly enhances the performance of DouDiZhu AI by combining DRL with MCTS. It outperforms rule-based heuristics with an Average Difference in Points (ADP) of 0.72 and achieves human-level performance, as evidenced by its ADP of −0.07 when compared to top human players. Moreover, by incorporating Deep Neural Networks (DNN) into traditional Monte-Carlo methods, Zha et al. [13] propose the Deep Monte-Carlo (DMC) method, which effectively addresses the problems in DouDiZhu. It surpass DeltaDou with an ADP of 0.258, establishing itself as the most advanced DouDiZhu AI. Chow et al. [45] extend the DMC method to Big2, and the DMC agent surpass PPO-based agents with ADP of 1.204. The exceptional performance of DMC in DouDiZhu and Big2 highlights the effectiveness of Monte-Carlo methods in large-scale, complex card games, offering novel perspectives for future research on addressing issues such as complex action spaces, multiplayer, sparse rewards, and imperfect information. However, DeltaDou trains its model for two months using 68 CPUs, while DouZero trains its network with four GPUs and 48 CPUs for 30 days. The resource requirements and training time pose challenges for small-scale laboratories, individual researchers, and rapid deployment. A comparative analysis of previous literature is shown in Table 1. In this paper, we mitigate both the equipment requirements and training time, making them more accessible and affordable for a wider range of research labs and individual researchers.

3. Preliminary

3.1. DouDiZhu

DouDiZhu is a popular card game in China and is typically played by three players, each player is assigned a specific role: Landlord, Peasant Down, and Peasant Up. The game proceeds in a counterclockwise direction, with Landlord playing first, followed by Peasant Down (the player to the right of Landlord) and then Peasant Up (the player to the left of Landlord). DouDiZhu is played with a deck of 54 cards, which consists of 15 different ranks: 3, 4, 5, 6, 7, 8, 9, 10 (T), J, Q, K, A, 2, black joker (B), and red joker (R). These ranks are ranked from lowest to highest. Each rank has four cards, except for the black joker and red joker, which come in four suits: heart, spade, club, and diamond. Suits are irrelevant.

At the start of the game, each player receives 17 private cards. The final three cards are placed face-down and are visible to all players until the bidding is over. Players can bid 1, 2, 3, or choose to "Pass" (not bid). The player who bids the highest wins the three public cards. Once the bidding round is over, the highest bidder becomes Landlord, while the other two players are referred to as Peasant Down and Peasant Up. After that, the three face-down cards are revealed to all players and are given to Landlord (who now has a total of 20 cards). If no one is interested in the three public cards, the game restarts.

The objective of DouDiZhu is to be the first player to play all of the cards in one's hand. If the Peasants are able to defeat the Landlord, they will receive a reward from the Landlord. In this case, both Peasants are equal winners and receive the same reward. Therefore, the Peasants may work together to try to defeat the Landlord. If the Landlord plays all of his/her cards, he/she is the winner and will receive a reward from each Peasant. A detailed introduction is provided in Appendix A.

3.2. Big2

Big2 (also known as Chor Dai Di or Deuces) is a four-player card game of Chinese origin which is played widely throughout East and South East Asia. The object of the game is to be the first to get rid of all of a player's cards. Cards can be played singly or in certain combinations. If a player cannot be first to play all his/her cards, then his/her aim is to have as few cards as possible when another player wins.

Big2 presents a unique challenge for reinforcement learning because of the long-term planning required to balance the trade-off between flexibility and risk in the late game. The actions taken in the early game are highly correlated with the available options in the late game, making it critical for the agent to accurately assess the expected return and take appropriate actions. The penalty for losing increases with the number of unplayed cards, which can result in a significant penalty in the later stages of the game. Unlike DouDiZhu, where the penalty for losing and winning is fixed, Big2 requires a more nuanced approach to maximize the chances of winning. Balancing the need for maneuverability in the later stages of the game with the risk of a high penalty is the key challenge that reinforcement learning agents must address when playing Big2.

Furthermore, the nature of collaboration between agents in Big2 is inherently fluid and temporary. Specifically, players pass their turn when the opponent before them has just played, in order to preserve high-ranking cards for later and potentially gain the opportunity to play second. Conversely, players tend not to pass if the player after them has played, in order to avoid the risk of playing last on the subsequent round. However, each agent's strategic decision-making process is influenced by the positions of their competitors. For example, an agent may choose not to give the upstream-player a chance to pass if they have fewer cards and are more likely to win if they take the lead. Therefore, the nature of collaboration among agents in Big2 is highly dynamic and subject to constant adaptation based on the ever-changing context of the game. More details are provided in Appendix B.

3.3. DouZero

DouZero [13] is the state-of-the-art artificial intelligence (AI) for the challenging card game DouDiZhu, and it uses the Deep Monte-Carlo (DMC) method. DMC is a model-free and value-based reinforcement learning method, representing a variant of the Monte-Carlo (MC) method that utilizes deep neural networks for function approximation. In DMC, a Q-network is utilized to estimate $Q(s, a)$, where both state s and action a are concatenated as input, and the mean-square-error (MSE) loss is used for parameter updates. DMC samples a batch of episodes and optimizes its Q-network using all instances (s, a, r) for

Table 1
Summary of related work.

Reference	Advantages	Disadvantages
CFR [15]	CFR is well-suited for games like heads-up limit hold 'em poker where players have imperfect information. It can handle large game trees, which is crucial in complex games.	CFR requires significant computational resources, especially for very large game trees. Storing the regret values for each state–action pair can be memory-intensive.
MCCFR [18]	By using sampling methods, MCCFR reduces the computational intensity compared to standard CFR. It is more efficient in handling extremely large game trees.	MCCFR faces challenges in solving complex card games due to their domain-specific rules and gameplay mechanics. Choosing and implementing an effective sampling strategy can be complex.
MCTS [16]	MCTS is applicable to a wide range of games and problems, including those with large or unknown search spaces. It does not require domain-specific knowledge, making it versatile and easily adaptable.	MCTS faces challenges in games with imperfect information due to unobserved data from other players. It can be resource-intensive, especially as the search space grows.
ISMCTS and its variants [22–27]	ISMCTS is designed to handle games where players have imperfect information. It uses a single tree for all players, simplifying the tree structure in multi-player games.	ISMCTS struggles to accurately estimate game states due to unobserved information, like hidden opponent cards. It suffers from strategy fusion since the same information set may correspond to different combinations of cards.
General Opponent Model [28]	By anticipating opponents' moves, the agent can play more strategically. In games with imperfect information, understanding an opponent's strategy can lead to better decision-making.	Opponent Model's accuracy is impacted by limited perfect information, especially at a card game's outset.
Bayesian Opponent Model and its variants [29–31]	Bayesian models can dynamically adapt to changes in an opponent's strategy over time.	Bayesian models often require prior knowledge or assumptions about opponents, which might not always be available or accurate.
PPO [13,38,40]	PPO offers more stable and reliable training of policies. It performs well in environments with continuous or discrete action spaces.	PPO, DQN, Actor–Critic, and CQL algorithms are susceptible to overestimation or failure to converge in complex card games due to sparse rewards and large action spaces. These algorithms show poor performance in DouDiZhu and Big2.
DQN [13,41]	DQN is relatively straightforward to implement and understand. It is particularly effective in environments with low-dimensional discrete action spaces.	
Actor–Critic [13,42]	Actor–Critic methods are flexible and can be applied to both discrete and continuous action spaces.	
CQL [43]	CQL applies DQN to high-dimensional discrete action environments through a two-stage DQN.	
DeltaDou [44]	DeltaDou is an AlphaZero-like deep reinforcement learning framework. It combines neural networks with MCTS and the inference algorithm in a self-play procedure.	The resource requirements and training time pose challenges for small-scale laboratories, individual researchers, and rapid deployment.
DMC [13,45]	DouZero demonstrates high-level performance in DouDiZhu, outperforming previous AI models and even strong human players.	

every-visit MC, where r is the reward. DMC predicts the discounted final reward r as there is only one nonzero reward at the terminal step of all episodes. The procedure of DMC is described in Algorithm 1.

DouZero's network input consists of a state feature and an action feature. The state feature represents perfect information, and the action feature describes the legal move corresponding to the current state. The state feature contains the player's cards, the union of other players' cards, the most recent actions, some one-hot vectors representing the number of bombs played and the number of cards currently held by each player, and historical moves. The action feature is encoded as a one-hot card matrix. Cards in both state and action features are encoded into a 4×15 one-hot card matrix, where rows represent the number of cards of each rank, and columns represent the fifteen ranks. DouZero's network architecture consists of an LSTM layer and a six-layer MLP. The LSTM layer receives historical moves as input, and its output is concatenated with the action feature and the remaining state feature.

DouZero parallelizes the DMC method with multiple actor processes and one learner process through self-play in a distributed training system, where actors play games to generate samples, and the learner trains the network based on these samples. Each actor maintains a local network for each agent and synchronizes it periodically with the global network. On the other hand, the learner maintains a global network for each agent and updates them based on the samples generated by the actor processes. More details about DouZero can be referred to in [13].

Algorithm 1 Deep Monte-Carlo Method.

Input: a large number of episodes $max\_episodes \rightarrow \infty$
procedure DMC($max\_episodes$)
1. Generate an episode using π by iterating over each legal action a_i in the action set A.
2. Update $Q(s,a)$ for each visited state–action pair with average return.
3. Update policy for each state s as $\pi(s) \leftarrow argmax_a Q(s,a)$.
4. Repeat (1) to (3) for a large number of episodes or forever to cover all states and actions.
end procedure

3.4. Extensive-form games

Extensive-form games are a model of strategic interaction between multiple agents in a sequential order. The representation is derived from a game tree and comprises the following components:

- : $\mathcal{N} = \{1, 2, \ldots, n\}$ is the set of agents;
- : $\mathcal{S}$ is the state space of the game;
- : $\mathcal{A}$ is the action space of the game;
- : p is a state transition probability defined as $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$;
- : r is a reward defined as $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R} \in \mathbb{R}$;

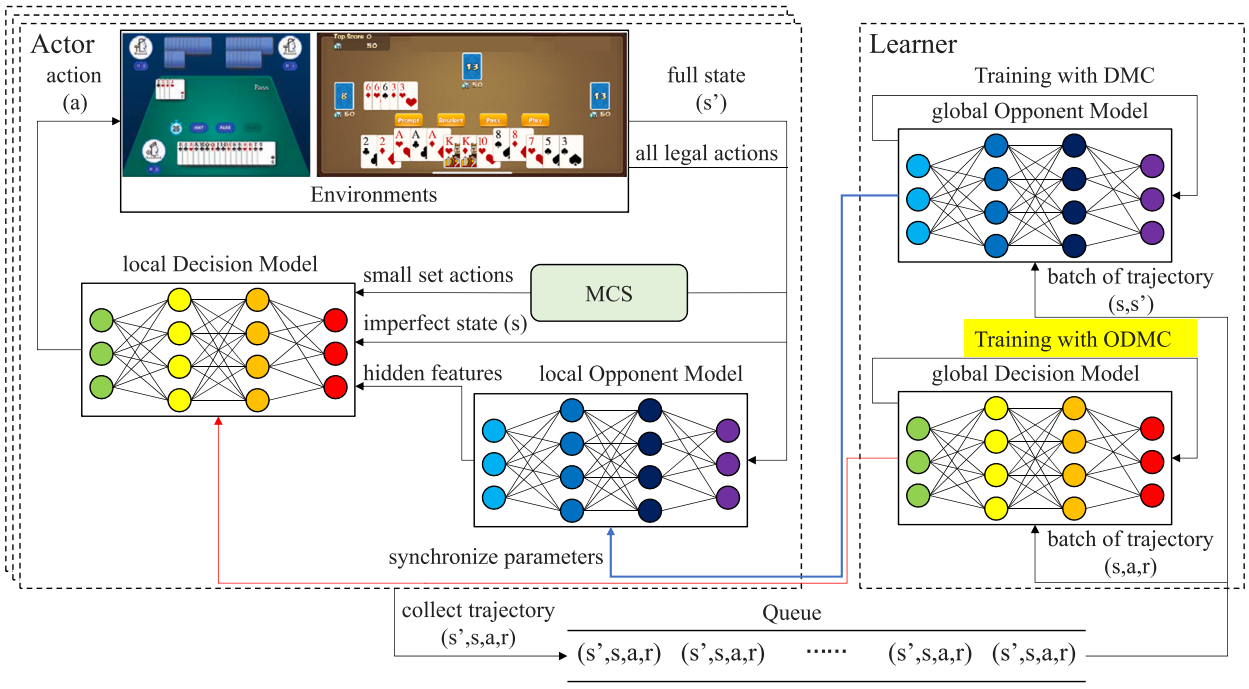


Fig. 1. The Optimized Deep Monte-Carlo Method (ODMC) with the opponent model and decision model.

- : $\gamma \in [0, 1]$ is the discount factor.

Extensive-form games, often represented as a 6-tuple $(\mathcal{N}, \mathcal{S}, \mathcal{A}, p, r, \gamma)$, serve as a foundational model. Within this framework, games like DouDiZhu or Big2 enable the execution of policy π on this environment to generate a sequence of states, actions, and rewards until the end of the game at timestep T: $s_0, a_0, r_1, \ldots, s_t, a_t, r_{t+1}, \ldots, s_{T-1}, a_{T-1}, r_T$. This sequence is called an episode. Formally, at each timestep t, the agent takes an action $a_t \in \mathcal{A}$ at state $s_t \in \mathcal{S}$ following a deterministic policy $\pi(s_t) = a_t$, transfer to the next state s_{t+1} with a probability $p(s_{t+1}|s_t, a_t)$, and receives a immediate reward $r_{t+1} = \mathcal{R}(s_t, a_t) \in \mathbb{R}$.

4. Method

The proposed approach involves first calculating the minimum combination to win the game from the cards in hand (as outlined in Section 4.1). Subsequently, the minimum combination and the other observed information are used to train an opponent model to predict the opponent's minimum combination and unobserved information such as hidden cards (as described in Section 4.2). The predicted minimum combination, along with the imperfect information from the opponent model, is then incorporated as input features to train the model using the Optimized Deep Monte-Carlo (ODMC) method. Additionally, the minimum combination is used to optimize and filter low-winning moves for each state, which reduces the vast space of actions sampled and estimated by Deep Monte-Carlo Method (DMC) (as explained in Section 4.3). The framework is shown in Fig. 1, which combines the Optimized Deep Monte-Carlo Method with the opponent model and decision model.

4.1. Minimum combination search

The minimum combination represents the fewest number of actions needed to play out all the cards held by a player according to the game rules. Based on the rules of DouDiZhu or Big2, the winner is the first player to clear all cards in their hand. As a result, it is reasonable for players to strategically split their cards into subsets within the legal action categories (refer to Appendix Table 12 and Appendix Table 13) to play all of their cards as quickly as possible.

MCS involves two key steps. Firstly, it requires identifying all possible variable-length (e.g., Chain, Pair Chain, and Plane in Appendix Table 12) or variable-suit (e.g., Straight, Flush, and Full House in Appendix Table 13) actions that can be formed. Secondly, it involves considering the remaining cards in hand as fixed-length or fixed-suit actions (e.g., Solo, Pair, and Triplet in Appendix Tables 12 and 13). Variable-length or variable-suit actions are characterized by their ability to change in length or suit depending on the specific cards involved. Conversely, fixed-length or fixed-suit actions involve playing cards that do not allow for variation in their required number or suit of cards. The minimum number of combinations, denoted by K, can be expressed as (1), with respect to a given set of cards in hand, denoted as X.

$$K = \min_{x \subseteq X}(Variable(x), Fixed[a][b][m][n]) \tag{1}$$

Here, x represents the set of cards that comprise variable-length or variable-suit actions, $Variable(x)$ represents the minimum number of combinations required for variable-length or variable-suit actions, and $Fixed[a][b][m][n]$ represents the minimum number of combinations necessary for fixed-length or fixed-suit actions. The variables a, b, m, and n respectively denote the number of Bombs, Triplets, Pairs, and Solos present in the set of cards.

By following these two steps, MCS determines the minimum number of card combinations required to win the game. It is important to note that the implementation of this method may vary depending on the specific game rules. The MCS algorithm for Big2 is presented in Algorithm 2, and for DouDiZhu, it can be found in Algorithm 5 of Appendix D.

4.2. Opponent model

Opponent model involves constructing a model of the opponent to predict various properties of the opponent's behavior, such as their actions, preferences, and unobservable information. In complex imperfect-information games, agents often lack complete knowledge of the game state, which hinders their learning progress. However, by employing opponent model, the agents can predict the likely state of their opponents, even when faced with imperfect information.

Algorithm 2 Minimum Combination Search for Big2.

Input: The quantity of 13 different rank cards: $x[0...12]$
Output: The minimum combinations K for x
procedure $Variable(x)$

1. Iterate over the elements in x with three for-loops, to find all possible variable-suit actions:
   1.1 If any **Straight** $\in x$, create a copy of x, denoted as $x^{'}$, and remove the same number of identical elements from $x^{'}$ as this **Straight**. Then call $Variable(x^{'})$ recursively to obtain the minimum value $K = min(K, Variable(x^{'}) + 1)$ until the end of the iteration.
   1.2 If any **Flush** $\in x$: ▷ Straight Flush is a special subcategory of Flush
      1.2.1 Create a copy of x, denoted as $x^{'}$, and remove the same number of identical elements from $x^{'}$ as this **Flush**. Then, call $Variable(x^{'})$ recursively to obtain the minimum value $K = min(K, Variable(x^{'}) + 1)$ until the end of the iteration.
2. Iterate over x to count the remaining number of **Bomb**, **Triplet**, **Pair**, and **Solo** denoted as a, b, y, and z, respectively. Here, **Bomb** refers to four cards with the same rank but different suits. After counting the cards, return the minimum value between K and $Fixed[a][b][y][z]$.

end procedure
procedure $Calculate\_Fixed(a, b, y, z)$
$Fixed[a][b][y][z] \leftarrow \min($
$Fixed[a][b][y][z]$, ▷ Current minimum value
$Fixed[a][b][y][z-1]+1$, ▷ Play a Solo
$Fixed[a][b][y-1][z]+1$, ▷ Play a Pair
$Fixed[a][b][y-1][z+2]$, ▷ Split a Pair to two Solos
$Fixed[a][b-1][y][z]+1$, ▷ Play a Triplet
$Fixed[a][b-1][y-1][z]+1$, ▷ Play a Full House
$Fixed[a][b-1][y+1][z+1]$, ▷ Split a Triplet to a Pair and a Solo
$Fixed[a][b-1][y][z+3]$, ▷ Split a Triplet to three Solos
$Fixed[a-1][b][y][z-1]+1$, ▷ Play a Quads
$Fixed[a-1][b+1][y][z+1]$, ▷ Split a Bomb to a Triplet and a Solo
$Fixed[a-1][b][y+2][z]$, ▷ Split a Bomb to two Pairs
$Fixed[a-1][b][y+1][z+2]$, ▷ Split a Bomb to a Pair and two Solos
$Fixed[a-1][b][y][z+4])$ ▷ Split a Bomb to four Solos
return $Fixed[a][b][y][z]$
end procedure
procedure MAIN

1. $Fixed[a][b][y][z]$ is preprocessed beforehand, then it can be directly queried in the $Variable(x)$ function.
2. $K \leftarrow Variable(x)$

end procedure

The proposed opponent model, as shown in Fig. 2(b), predicts the hidden cards held by the other player and the minimum possible card combinations in their hand. We use a single deep neural network to predict the hidden information for all players, and its architecture is listed in Table 2. The output of the opponent model includes the probabilities of each card (a set of 54 1×1 vectors) and the probabilities of the number of minimum combinations (a 20×1 vector). Essentially, the opponent model is a multi-head classifier that generates a probability distribution for each possible card and the minimum possible combinations of cards in the other players' hand. By adopting an oracle view, unobservable information, namely the specific cards held by each player, can be observed. Each card is encoded with a one-hot 1×1 label, which is a binary vector where every card is represented by a unique bit position. A value of 1 indicates the presence of the corresponding card, while 0 indicates its absence. One-hot labels can be used for opponent model training, allowing the model to learn patterns and make predictions based on the presence or absence of a certain card. These one-hot labels include information about the presence of cards such as the ♢3 or the ♣3, as well as the minimum combinations of the cards. As these labels are obtained from either the oracle view or the proposed MCS, they can correct the output predicted by the opponent model.

Table 2
Architecture of opponent model.

Opponent model	Input	Output
1st layer in MLP	[batch,1052]	[batch,1024]
2nd layer in MLP	[batch,1024]	[batch,512]
3rd layer in MLP	[batch,512]	[batch,512]
4th layer in MLP	[batch,512]	[batch,512]
5th layer in MLP	[batch,512]	[batch,512]
FC1,FC2, …,FC54	[batch,512]	[batch,1]
FC56, …,FC110		
FC112, …,FC166		
FC55,FC111,FC167	[batch,512]	[batch,20]

Table 3
Architecture of decision model.

Decision model	Input	Output
LSTM	[batch,4,216]	[batch,4,128]
1st layer in MLP	[batch,1402]	[batch,1024]
2nd layer in MLP	[batch,1024]	[batch,512]
3rd layer in MLP	[batch,512]	[batch,512]
4th layer in MLP	[batch,512]	[batch,512]
5th layer in MLP	[batch,512]	[batch,512]
6th layer in MLP	[batch,512]	[batch,1]

The decision model is enhanced by integrating the opponent model into DouZero's model, as shown in Fig. 2(a). The decision model consists of an Long Short-Term Memory (LSTM) with four hidden units, each having 128 time steps, and a six-layer Multilayer Perceptron (MLP). The architecture of the decision model is provided in Table 3. The LSTM layer encodes the past actions (4×216) and incorporates the output (4×128) as part of the state features. The state features (1×1052), action features (1×54), and hidden features (1×296) predicted by the opponent model are concatenated and fed into the MLP of the decision model. These features are described in Table 14 of Appendix C, where the decision model and opponent model share state features, and the opponent model's output is used as part of the decision model's input features.

4.3. Optimized deep Monte-Carlo method

Algorithm 1 outlines the DMC method, which utilizes a neural network to evaluate the state–action value function $Q(s, a)$ and updates it using mean-square-error (MSE). One important implementation detail when structuring a neural network to approximate the $Q(s, a)$ is to input a state and a legal action, and output a single value instead of all Q-values for a given state at once, as shown in Fig. 2(a). This approach is particularly useful for games like DouDiZhu and Big2, which have a large action space and sparse legal actions in each state. However, evaluating all possible actions in each state can be computationally expensive, especially for games with large state and action spaces. The objective of both games is to be the first player to empty their hand, which requires minimizing the number of times cards are played by aiming for the smallest possible combination. However, when the opponent has only a few cards left or is about to run out of all their cards in several rounds, it is necessary for the other team or player

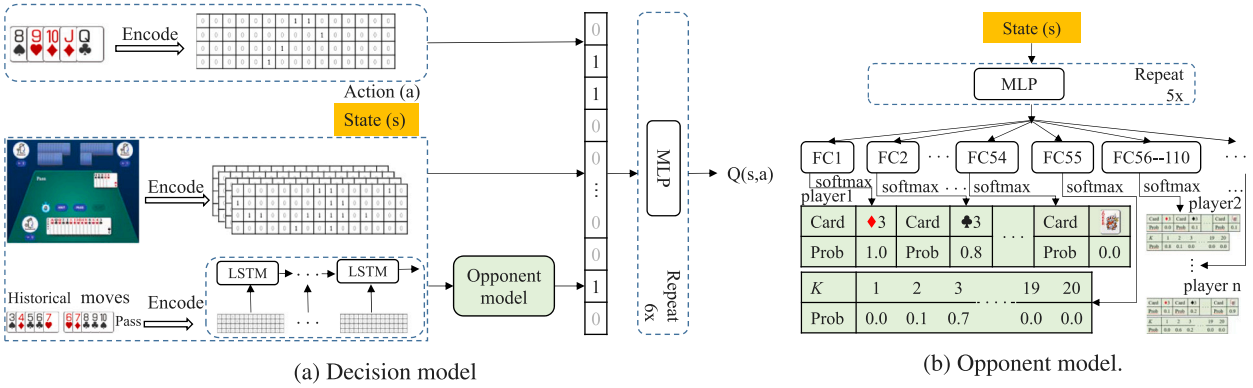


Fig. 2. (a) The decision model based on DouZero is enhanced with an opponent model. (b) The opponent model takes state features as input, and outputs the probabilities of each card and the minimum combinations of other players. The action value $Q(s,a)$ is trained using DMC. The prediction of the cards and the minimum combinations K of other players are concatenated with the state features and action features. All these features are then fed into the decision model to determine which action to take. The opponent model can be viewed as a multi-head classifier, which includes a layer of LSTM to encode historical moves, five shared layers of MLP, and multi-head Fully Connected (FC) layers to output the probabilities.

to attempt more legal actions to prevent the opponent from winning the game. To address this problem, a modified version of Deep Monte-Carlo (DMC) is proposed, named Optimized Deep Monte-Carlo, which involves pruning the set of legal actions A for each state s to generate a smaller set of actions A' that can be sampled in Step 1 of Algorithm 1, using the pruning (2).

$$\begin{aligned} &l = \mid A \mid \\ &K_{min} = min(K_1, \ldots, K_i, \ldots, K_l) \\ &A' = A' \cup a_i, \quad a_i \in A,\ K_i \le K_{min} + \frac{N}{n} + d, n \ge 1, d \ge 1 \end{aligned} \tag{2}$$

where K_i is the minimum combination number after playing action a_i, which is calculated by (1). N denotes the average initial number of cards per player, n refers to the fewest number of cards held by any player, and d indicates the current round of the game. (2) reveals that as the number of game rounds increases and the number of cards held by the player decreases, the scope of action pruning will gradually expand. This helps reduce the action space in the early stages and ensures a higher chance of winning in the middle and later stages of the game.

Algorithm 3 Asynchronous Parallel Actor.

Input: shared memory M, exploration hyperparameter ϵ
Initialize N Actor Q-networks $\theta_1, \theta_2, ..., \theta_N$
Initialize N Actor cache $C_1, C_2, ..., C_N$
for each Actor **do Asynchronously**
Pull parameters from the global Q-network, set $\theta_i \leftarrow \theta_G$
Collect trajectories based on performing Equation (2):

$$a_t = \begin{cases} argmax_a Q_i(s_t, a_i \in A'), & with\ prob\ (1-\epsilon) \\ random\ Action & with\ prob\ \epsilon \end{cases}$$

Push trajectories(s_t, a_t, r_{t+1}) into C_i, if the amount of trajectories in C_i exceeds threshold D, then move trajectories size of D to M for the global Q-network to train.
end for

5. Experiments

In this section, we describe the experiments conducted and present the results obtained to demonstrate the improved learning efficiency achieved with the proposed methods in two card games: DouDiZhu and Big2. We first describe the experiment setup (refer to Section 5.1). Subsequently, we present an ablation experiment for the proposed opponent model and Optimized Deep Monte-Carlo Method (ODMC) (refer to Sections 5.2 and 5.3). Finally, we combine these two approaches and provide a comprehensive evaluation to demonstrate their combined effectiveness in improving learning efficiency (refer to Sections 5.4 and 5.5). DouZero (https://github.com/kwai/DouZero) and BigZero (https://github.com/johnnyhoichuen/big2-rl) are used as benchmark opponents for DouDiZhu and Big2, respectively.

Algorithm 4 Asynchronous Learner.

Input: shared memory M, batch size B, learning rate α
Initialize global Q-network θ_G
for $1 \rightarrow \infty$ **do**
Wait for all Actor trajectories in M, until the amount of trajectories in M exceeds B
Compute gradient Δ_{θ_G} using B trajectories in M.
Update the global Q-network $\theta_G \leftarrow \theta_G + \alpha\Delta_{\theta_G}$ with MSE loss and learning rate α
end for

5.1. Experiment setup

We assess the learning efficiency of the proposed approach in DouDiZhu and Big2 using the same methodology employed in DouZero and [45]. The latter is a benchmark that applies DouZero's method to Big2, referred to as BigZero. The strength of the initial hand cards, which is highly dependent on luck, poses a threat to the validity of the experiments. To mitigate this threat and ensure an unbiased and fair assessment, 10,000 decks are randomly dealt to two competing algorithms, A and B, with each algorithm playing as opposing camps. Subsequently, sides are switched, and the same deck is replayed. In DouDiZhu, agent A and B play as Landlord and Peasants once, respectively. In Big2, agent A play from each of the four positions to avoid positional bias, while agent B play in the remaining positions, ensuring an impartial assessment, as certain players may have significantly stronger hands than others in some games. In order to evaluate the learning efficiency improvement, we conduct tests on the model every 30 min on a set of 10,000 games to assess its performance during the training process. Furthermore, for a fair comparison, the original DouZero and BigZero models are trained on the same server for the same duration (30 days), referred to as DouZero* and BigZero* respectively. Their performances are compared with the proposed algorithm.

The evaluation metrics used to assess DouDiZhu include Winning Percentage (WP) and Average Difference in Points (ADP). WP represents the ratio of games won by algorithm A to the total number of games, while ADP denotes the average score per game between

Table 4
Comparison between D+OMODMC, DouZero, D+ODMC, D+OM, DouZero*.

Rank	B / A	D+OMODMC		DouZero [13]		D+ODMC		D+OM		DouZero*	
		mean	interval	mean	interval	mean	interval	mean	interval	mean	interval
1	D+OMODMC	–	–	0.07	0.01, 0.13	0.11	0.04, 0.18	0.22	0.17, 0.27	0.35	0.26, 0.43
2	DouZero	−0.07	−0.13, −0.01	–	–	0.04	−0.03, 0.11	0.14	0.07, 0.21	0.29	0.23, 0.35
3	D+ODMC	−0.11	−0.18, −0.04	−0.04	−0.11, 0.03	–	–	0.14	0.09, 0.19	0.26	0.17, 0.35
4	D+OM	−0.22	−0.27, −0.17	−0.14	−0.21, −0.07	−0.14	−0.19, −0.09	–	–	0.13	0.05, 0.21
5	DouZero*	−0.35	−0.43, −0.26	−0.29	−0.35, −0.23	−0.26	−0.35, −0.17	−0.13	−0.21, −0.05	–	–

The 'mean' indicates the average of the samples, and 'interval' indicates the 95% confidence bounds of the T-test.
D+OMODMC, D+ODMC, D+OM, and DouZero* are trained on the same server with one GPU for 30 days, while DouZero is trained on four GPUs for the same duration. The algorithms are ranked based on the ADP of other algorithms they outperform. Algorithm A outperforms B if the confidence interval of ADP is larger than zero.

algorithms A and B. These evaluation metrics can also be used to define rewards. Specifically, for WP, the agent receives a +1 reward if it wins a game and a −1 reward if it loses, while ADP can be used directly as a reward in place of WP. The metrics utilized for Big2 by BigZero and [22] are losing points and mean episode return (EV). However, losing points, EV, and ADP are equivalent metrics. Therefore, ADP serves as the unified evaluation metric for both games in this study. The agent are trained with ADP as the objective, and their performance is compared to variants of themselves and state-of-the-art baselines.

The experimental setup keeps the same hyperparameters and settings as DouZero [13]. For instance, in Algorithm 1, the number of episodes $max\_episodes$ is set to 100,000,000,000. In Algorithm 3, the exploration factor ϵ is fixed at 0.01. Additionally, Algorithm 4 maintains a batch size B of 32 and a learning rate α of 0.0001. These settings remain consistent as the proposed method builds upon and enhances the foundation established by DouZero. The DouDiZhu and Big2 agents are trained on two servers, each equipped with four Intel(R) Xeon(R) CPU E5-1620 v4 @ 3.50 GHz and a GeForce GTX 1080 GPU.

5.2. Evaluation of opponent model

The proposed opponent model involves the use of deep neural networks to predict the cards held by the other player and their minimum possible card combinations. The approach is discussed in detail in Section 4.2. Two experiments are conducted to evaluate the impact of integrating opponent model into the DouZero and BigZero on the learning efficiency of the original systems. The first compared the original DouZero* with the improved version incorporating opponent model (D+OM), while the second compared the original BigZero* with the improved version using opponent model (B+OM). DouZero* and D+OM are tested using the DouZero baseline, while BigZero* and B+OM are tested using the BigZero baseline. The results of these experiments are presented in Fig. 3, respectively. It can observe that when compared to the original versions, D+OM and B+OM do not show significant advantages in ADP in the earlier five days training. This can be attributed to the randomization of neural network parameters and the insufficient training time to yield substantial improvements in results. However, after 10 days of training, the benefit of applying opponent model in DouZero and BigZero became evident. It can be observed that there is a gradual increase in ADP for D+OM compared to DouZero* and B+OM compared to BigZero*. After training for the same duration (30 days), the ADP of D+OM is 0.13 higher than DouZero*, and the ADP of B+OM is 0.21 higher than BigZero*, as shown in Tables 4 and 5. The results demonstrate that the Decision Model, depicted in Fig. 2(a), effectively enhances learning efficiency by integrating the Opponent Model, shown in Fig. 2(b). One possible explanation is that the proposed opponent model predicts each player's imperfect information, thus reducing the impact of uncertainties during training.

5.3. Evaluation of Optimized Deep Monte-Carlo method

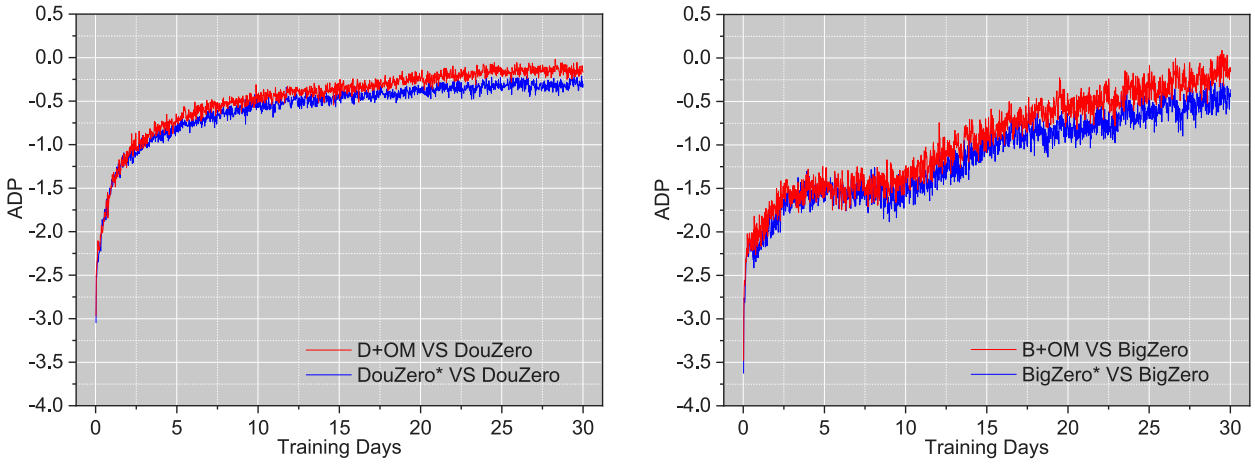
Optimized Deep Monte-Carlo (ODMC) is an enhanced version of the Deep Monte-Carlo method (DMC), which involves pruning the set of legal actions for each state to generate a smaller set of actions that can be sampled. This modification is discussed in detail in Section 4.3. The effectiveness of ODMC is evaluated by applying it to DouDiZhu (referred to as D+ODMC) and Big2 (referred to as B+ODMC) and comparing it with the original DMC method, which includes DouZero*, DouZero, BigZero*, and BigZero, respectively. The results of the experiments are shown in Fig. 4. During 30 days of training, it can be observed that D+ODMC and B+ODMC achieve an ADP close to zero, while DouZero* and BigZero* do not. To investigate potential reasons, an analysis is conducted on the pruning ratio when applying the Minimum Combination Search (MCS). Specifically, the number of valid actions and the number of valid actions are recorded after MCS pruning over 1,000,000 times of card playing with different values of $N/n+d$ in (2). Then, the pruning ratio is calculated for DouDiZhu and Big2, as shown in Tables 6 and 7, respectively. The results indicate that the pruning ratio is approximately 10% during the initial two rounds, enabling to achieve higher learning efficiency throughout the game training. Consequently, the proposed pruning technique resulted in a reduction in the number of legal actions, resulting in faster convergence and maintaining an advantage of 0.26 and 0.34 in ADP for D+ODMC and B+ODMC compared to DouZero* and BigZero*, respectively, as shown in Tables 4 and 5. However, after 25 days of training, achieving further performance improvement becomes challenging. One potential explanation for this is that MCS does not filter legal actions beyond the 6th round of play (pruning ratio ≈ 0) and $K_i \geq K_{min} + N/n + d$ in (2). Overall, the results demonstrate the effectiveness of ODMC in improving learning efficiency in the card game DouDiZhu and Big2.

5.4. Evaluation of combination of two methods

The aforementioned experiments have demonstrated that both of the proposed methods improve the learning efficiency of DouZero and BigZero. In this section, the evaluation focuses on the combination of these two methods: Opponent Model and Optimized Deep Monte-Carlo. These two methods are applied to DouDiZhu (D+OMODMC) and Big2 (B+OMODMC) and compared with the original method (DouZero and BigZero). The results of this combination are shown in Fig. 5. For a comprehensive evaluation, Optimized Deep Monte-Carlo (D+ODMC and B+ODMC) and the opponent model (D+OM and B+OM) are included to Fig. 5. It can be observed that D+OMODMC is superior to D+ODMC and D+OM, while B+OMODMC also outperforms B+ODMC and B+OM in terms of performance and learning efficiency. A comparison among these agents demonstrates that D+OMODMC and B+OMODMC rank first in DouDiZhu and Big2, as shown in Tables 4 and 5. These results demonstrate that the integration of an opponent model and Optimized Deep Monte-Carlo significantly enhances learning efficiency and performance.

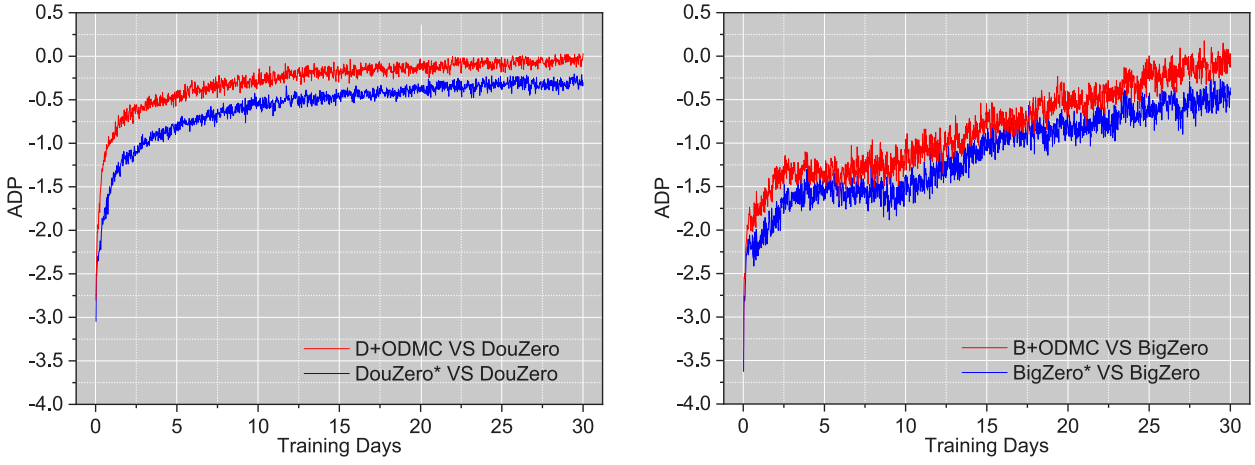
5.5. Evaluating with state-of-the-art baseline

The ablation evaluation above demonstrates that both the proposed Opponent Model and Optimized Deep Monte-Carlo method improve the



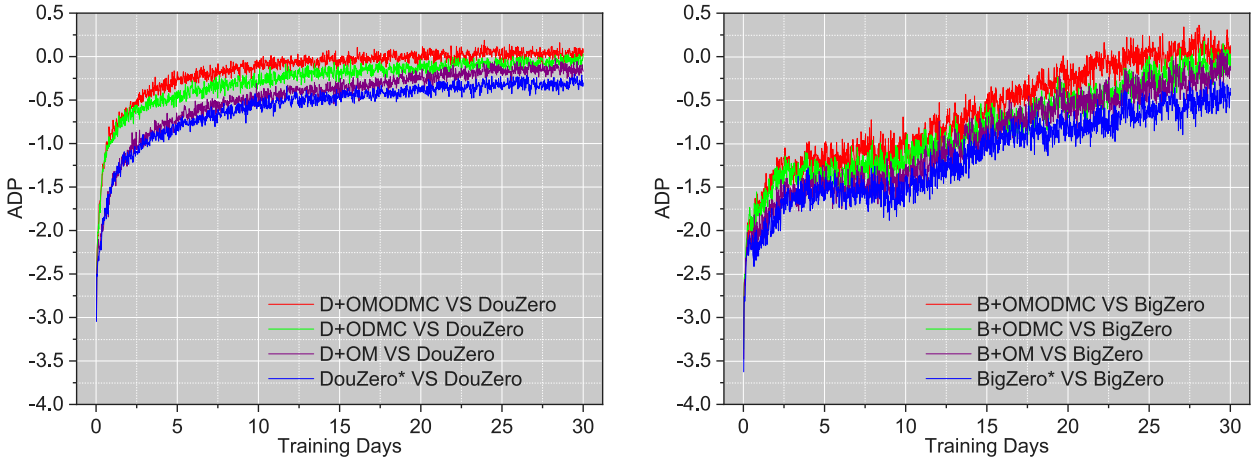
(a) Opponent model for DouDiZhu (b) Opponent model for Big2

Fig. 3. DouZero* and BigZero* are the original versions for DouDiZhu and Big2, while D+OM and B+OM are their respective improved versions with an opponent model.



(a) Optimized Deep Monte-Carlo for DouDiZhu. (b) Optimized Deep Monte-Carlo for Big2.

Fig. 4. DouZero* and BigZero* are the original versions for DouDiZhu and Big2, while D+ODMC and B+ODMC are their respective improved versions with Optimized Deep Monte-Carlo.



(a) Combination for DouDiZhu. (b) Combination for Big2.

Fig. 5. DouZero* and BigZero* are the original versions for DouDiZhu and Big2, while D+OMODMC and B+OMODMC are their respective improved versions with an opponent model and Optimized Deep Monte-Carlo.

Table 5
Comparison between B+OMODMC, BigZero, B+ODMC, B+OM, BigZero*.

Rank	B / A	B+OMODMC		BigZero [45]		B+ODMC		B+OM		BigZero*	
		mean	interval	mean	interval	mean	interval	mean	interval	mean	interval
1	B+OMODMC	–	–	0.14	0.03, 0.25	0.19	0.09, 0.29	0.21	0.10, 0.32	0.48	0.36, 0.60
2	BigZero	−0.14	−0.25, −0.03	–	–	0.02	−0.09, 0.13	0.12	0.01, 0.23	0.41	0.28, 0.54
3	B+ODMC	−0.19	−0.29, −0.09	−0.02	−0.13, 0.09	–	–	0.07	−0.04, 0.18	0.34	0.21, 0.47
4	B+OM	−0.21	−0.32, −0.10	−0.12	−0.23, −0.01	−0.07	−0.18, 0.04	–	–	0.21	0.11, 0.31
5	BigZero*	−0.48	−0.60, −0.36	−0.41	−0.54, −0.28	−0.34	−0.47, −0.21	−0.21	−0.31, −0.11	–	–

The 'mean' indicates the average of the samples, and 'interval' indicates the 95% confidence bounds of the T-test.
B+OMODMC, B+ODMC, B+OM, and BigZero* are each trained on the same server for 30 days, while BigZero is trained for 60 days. The algorithms are ranked based on the ADP of other algorithms they outperform. Algorithm A outperforms B if the confidence interval of ADP is larger than zero.

Table 6
Pruning ratio for DouDiZhu.

$\frac{N}{n}+d$	Num of valid actions	Num of actions after MCS	Pruning ratio
2	8,760,469	7,206,362	17.74%
3	8,813,328	7,532,752	14.53%
4	8,758,687	7,813,625	10.79%
5	8,773,557	8,239,247	6.09%
6	8,822,645	8,516,499	3.47%
7	8,668,932	8,655,062	0.16%

The pruning ratio is computed over 1,000,000 times of card playing, with different values of $N/n+d$. Pruning ratio = 1 −Number of actions after MCS ÷ Number of valid actions.

Table 7
Pruning ratio for Big2.

$\frac{N}{n}+d$	Num of valid actions	Num of actions after MCS	Pruning ratio
2	6,583,647	5,738,965	12.83%
3	6,594,792	5,941,248	9.91%
4	6,578,073	6,176,810	6.10%
5	6,569,788	6,388,461	2.76%
6	6,603,255	6,564,956	0.58%
7	6,597,609	6,595,629	0.03%

The pruning ratio is computed over 1,000,000 times of card playing, with different values of $N/n+d$. Pruning ratio = 1 −Number of actions after MCS ÷ Number of valid actions.

learning efficiency of DouZero and BigZero. To further evaluate the differences in learning efficiency and performance between the proposed approach and the existing state-of-the-art artificial intelligence (AI), a comparison is made between D+OM, D+ODMC, and D+OMODMC with the following DouDiZhu state-of-the-art baselines:

- **DouZero** [13]. It is the latest and strongest open-sourced DouDizhu AI system that utilizes the Deep Monte-Carlo method.
- **SL** [13]. This human expert-level AI is based on supervised learning and utilizes 226,230 expert matches collected from the top players in a commercial DouDiZhu game mobile app, resulting in the generation of 49,990,075 samples for supervised learning.
- **CQL** [43]. Combinational Q-Learning (CQL) is a program based on two stages of Deep Q-Learning, with the authors providing the open-sourced code and the pre-trained model.
- **RLCard** [46]. A open-source heuristic rule-based program.

Also, a comparison is made between B+OM, B+ODMC, and B+OMODMC with the following Big2 competing benchmarks:

- **BigZero** [45]. This AI applies the DouZero method to Big2 and achieves excellent performance.
- **Rule-Based** [47]. A heuristic algorithm based on Big2 rules.
- **PPO** [39]. This AI utilizes the Proximal Policy Optimization (PPO) algorithm to train a model and achieves performance comparable to that of amateur human players.
- **DQN**. We employ Deep Q-Learning (DQN) to train a model over a period of 30 days.

Table 8
The number of days it takes for D+OMODMC, D+ODMC, D+OM, and DouZero* to reach the ADP threshold of zero when compared with each DouDiZhu baseline algorithm.

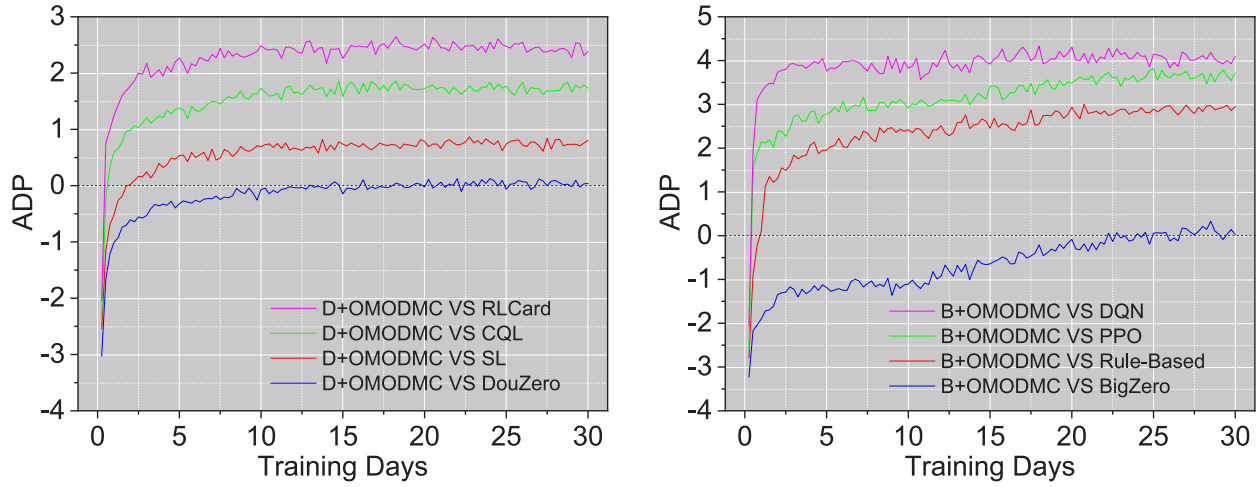
	DouZero	SL	CQL	RLCard
D+OMODMC	20.0	1.8	0.5	0.3
D+ODMC	29.5	2.4	0.7	0.4
D+OM	\	4.5	1.3	0.7
DouZero*	\	7.2	2.1	0.9

Time ratio=(1.8+0.5+0.3)/(7.2+2.1+0.9)*100% = 25.5%.

The results in Fig. 6(a) demonstrate that D+OMODMC outperforms SL, CQL, and RLCard in ADP, during short training periods. Similarly, the results in Fig. 6(b) reveal that B+OMODMC outperforms Rule-Based, PPO, and DQN in ADP, in the early stages of training. Despite D+OMODMC achieving a slight advantage over DouZero and B+OMODMC having a slight advantage over BigZero, it is noteworthy that DouZero uses four times as many GPUs as D+OMODMC, while BigZero requires twice the number of training days as B+OMODMC. For more specific timing information, Table 8 displays the number of days it takes for D+OMODMC, D+ODMC, D+OM, and DouZero* to reach the ADP threshold of zero when compared to each DouDiZhu baseline algorithm. Similarly, Table 9 presents the corresponding information for B+OMODMC, B+ODMC, B+OM, and BigZero* compared to each Big2 baseline algorithm. We obtain 10,000 respective samples generated by the optimal models of D+OMODMC, D+ODMC, D+OM, DouZero, and DouZero* when they compete against each baseline of DouDiZhu. Then, a significance T-test is conducted to compare the means and assess the 95% confidence interval of these observed samples, as shown in Table 10. Similarly, a significance T-test is performed to compare the means and evaluate the 95% confidence interval of the observed samples obtained from B+OMODMC, B+ODMC, B+OM, BigZero, and BigZero* when competing with each baseline of Big2, as shown in Table 11. If a confidence bound of ADP is greater than zero when comparing algorithm A with algorithm B, algorithm A is considered better than algorithm B. It can be observed that DouZero*, using the same code as DouZero, does not surpass DouZero even after 30 days of training. However, D+OMODMC slightly outperforms DouZero (with an ADP of 0.07) after being trained for 20.0 days. Similarly, BigZero*, using the same code as BigZero, fails to surpass BigZero even after 30 days of training, while B+OMODMC, trained for 25.0 days, exhibits slightly superior performance to BigZero (with an ADP of 0.14). Additionally, it can be noticed that the time required for D+OMODMC to surpass SL, CQL, and RLCard is only 25.5% of the time it took for DouZero*, and the time required for B+OMODMC to surpass Rule-Based, PPO, and DQN is also only 25.5% of the time it took for BigZero*. Moreover, D+ODMC, D+OM, B+ODMC, and B+OM require less time to beat the baseline algorithms compared to DouZero* and BigZero*. These findings strongly suggest that the proposed Opponent Model and Optimized Deep Monte-Carlo are highly effective in accelerating training.

6. Conclusion and future work

This work contributes to the theory of reinforcement learning by proposing a model that combines Deep Monte-Carlo (DMC) method



(a) Comparison of D+OMODMC and DouDiZhu baselines.

(b) Comparison of B+OMODMC and Big2 baselines.

Fig. 6. The ADP that D+OMODMC competes against in its training with DouZero, SL, CQL, and RLCard (left); The ADP that B+OMODMC fight against in its training with BigZero, Rule-Based, PPO, and DQN (right).

Table 9
The number of days it takes for B+OMODMC, B+ODMC, B+OM, and BigZero* to reach the ADP threshold of zero when compared with each Big2 baseline algorithm.

	BigZero	Rule-Based	PPO	DQN
B+OMODMC	25.0	0.8	0.3	0.2
B+ODMC	\	1.1	0.4	0.3
B+OM	\	2.2	0.7	0.5
BigZero*	\	3.4	0.9	0.8

Time ratio=(0.8+0.3+0.2)/(3.4+0.9+0.8)*100% = 25.5%.

with Minimum Combination Search (MCS) and opponent model to improve the learning efficiency of agents in complex imperfect-information card games such as DouDiZhu and Big2. The proposed method involves first calculating the minimum combination from the cards in hand and then training an opponent model that can predict the opponent's minimum combination and imperfect information. The predicted minimum combination, along with the imperfect information from the opponent model, is incorporated as a feature input in the model training process using the DMC method. Additionally, the minimum combination is used to optimize and filter low-winning moves for each state, which reduces the vast space of actions sampled and estimated by DMC. Due to the stochastic nature of card games, a modified variant of DMC, i.e., Optimized DMC (ODMC), is proposed to quantify the uncertainty involved in the model. The modification, in comparison with other novel approaches [48], considers pruning the set of legal actions for each state. The proposed framework is evaluated using DouDiZhu and Big2 games, and the results demonstrate that it learns faster than the original DouZero approaches, which represents our practical contribution.

Although the proposed methods improved learning efficiency, there are still limitations for further improvement. Firstly, our neural network architecture and basic method (DMC) remain the same as DouZero. We plan to explore other neural networks such as convolutional neural networks like Transformer [49], and other reinforcement learning methods to enhance the performance of the proposed framework. Secondly, MCS is limited by the rules of the game. However, it can be extended to other complex games by making minor modifications. For example, when extending MCS from Big2 to DouDiZhu, we modify Algorithm 2 to Algorithm 5. This modification can also serve as a reference when extending MCS to other card games, such as Contract Bridge [50] and Cuckoo [51]. Third, the accuracy of the opponent model is limited when the player holds many cards, especially in the first few rounds of a game. We will try to use all information obtained through unfair means, and then gradually replacing this observable information with the unobservable information predicted by our opponent model. Finally, the given results only uncover the samples and do not determine which variables have the most influence. Therefore, extraneous variables might interfere with the information, and thereby outcomes can be adversely impacted by the quality of the work. We will take into account sensitivity analysis [52] and model calibration for these variables through the optimal saved weight database.

CRediT authorship contribution statement

Qian Luo: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Tien-Ping Tan:** Writing – review & editing, Supervision, Resources, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This research is supported by the Fundamental Research Grant Scheme (FRGS/1/2021/ICT02/USM/02/4) Ministry of Higher Education, Malaysia.

Table 10
Tournament results between D+OMODMC, D+ODMC, D+OM and DouZero* with each baseline algorithm by playing 10,000 decks.

Rank		DouZero [13]		SL [13]		CQL [43]		RLCard [46]	
		mean	interval	mean	interval	mean	interval	mean	interval
1	D+OMODMC	0.07	0.01, 0.13	0.79	0.72, 0.86	1.76	1.69, 1.83	2.43	2.37, 2.49
2	**DouZero**	–	–	**0.70**	–	**1.69**	–	**2.29**	–
3	D+ODMC	−0.04	−0.13, 0.05	0.67	0.58, 0.76	1.65	1.57, 1.73	2.30	2.23, 2.37
4	D+OM	−0.14	−0.21, −0.07	0.62	0.54, 0.70	1.58	1.52, 1.65	2.22	2.15, 2.29
5	DouZero*	−0.29	−0.35, −0.23	0.57	0.49, 0.65	1.50	1.42, 1.58	2.16	2.09, 2.23

The 'mean' indicates the average of the samples, and 'interval' indicates the 95% confidence bounds of the T-test. Algorithm A outperforms B if the confidence interval of ADP is larger than zero.
The results highlighted in boldface are copied from DouZero [13].

Table 11
Tournament results between B+OMODMC, B+ODMC, B+OM and BigZero* with each baseline algorithm by playing 10,000 decks.

Rank	B / A	BigZero [45]		Rule-Based [47]		PPO [39]		DQN	
		mean	interval	mean	interval	mean	interval	mean	interval
1	B+OMODMC	0.14	0.03, 0.25	2.95	2.88, 3.03	3.72	3.65, 3.79	3.98	3.89, 4.07
2	BigZero	–	–	2.86	2.78, 2.95	3.67	3.60, 3.74	3.94	3.86, 4.02
3	B+ODMC	−0.02	−0.13, 0.09	2.84	2.75, 2.93	3.64	3.55, 3.73	3.91	3.80, 4.03
4	B+OM	−0.12	−0.23, −0.01	2.71	2.62, 2.80	3.55	3.47, 3.63	3.87	3.78, 3.96
5	BigZero*	−0.41	−0.54, −0.28	2.59	2.50, 2.68	3.52	3.43, 3.61	3.81	3.71, 3.91

The 'mean' indicates the average of the samples, and 'interval' indicates the 95% confidence bounds of the T-test. Algorithm A outperforms B if the confidence interval of ADP is larger than zero.

Appendix A. DouDiZhu

A.1. Rules

DouDiZhu consists of four distinct phases: Dealing, Bidding, Playing, and Scoring. Each of these phases is briefly described below.

- **Dealing:** A shuffled pack of 54 cards is dealt to the three players, with each player receiving 17 cards privately. The remaining three cards are kept face down on the deck for the next phase.
- **Bidding:** The three players take turns bidding and outbidding for the position of the Landlord based on the strength of their hand cards. Once the Landlord is determined, the remaining three cards are revealed and given to the Landlord, while the other two players become the Peasants.
- **Playing:** The playing phase is composed of multiple rounds, with the Landlord playing first in the initial round. Each player takes turns playing a legal action from "Action Category" column in Table 12, and the following player can choose to pass or play a higher-ranked action. If all players pass consecutively, the player who played the last hand starts the next round. The game continues until one player empties all their hand cards, with the Landlord winning if he/she has no cards left and the Peasant team winning if either peasant has no cards left. The Peasant team needs to cooperate to increase their chances of winning, and even a player with weak hand cards can contribute to the team's victory.
- **Scoring:** The game is over when one of the players plays all his/her cards. If the Landlord plays all his/her cards, he/she is the winner, but if one of the Peasants plays all his/her cards, both Peasants are equal winners. The base winning point is set to one and is multiplied by the multipliers that depend on the bid and whether the player is the Landlord or a Peasant. Each Rocket or Bomb will double the base winning point.

A.2. State and action space

DouDiZhu has a maximum of 10^{83} states, and while the size of its state space is not as extensive as that of No-limit Texas Hold'em, which can be as high as 10^{126}, acquiring an effective strategy is still a daunting task [46]. This is because agents must precisely differentiate between different states, as each card has a significant impact on the game's outcome. Therefore, the state space of DouDiZhu cannot be easily abstracted due to the potential for even minor variations in the state representation to significantly influence the strategy.

DouDiZhu is a complex card game that possesses a large action space due to the multitude of possible card combinations. The size of the action space of DouDiZhu is summarized in the "Num" column of Table 12 and amounts to 27,472. In contrast to No-limit Texas Hold'em, which features a manageable action space that can be easily abstracted, DouDiZhu is much more intricate. In DouDiZhu, the importance of each card in determining the optimal action cannot be overstated. For instance, in action Triplet with Solo, selecting an inappropriate kicker may result in a loss due to the potential for breaking a Chain. Consequently, abstracting the action space of DouDiZhu poses a considerable challenge. This presents a significant problem for reinforcement learning algorithms, as most of them demonstrate superior performance only on small action spaces.

Appendix B. Big2

B.1. Rules

Big2 consists of three distinct phases: Dealing, Playing, and Scoring. Each of these phases is briefly described below.

- **Dealing:** At the beginning of the game, each player is dealt 13 cards. The player with ◇3 goes first, and the game proceeds clockwise.
- **Playing:** Players take turns playing actions from the "Action Category" column in Table 13, and each action played must be of a higher rank than the previous one. The first player to play all their cards wins the round, and the remaining players receive penalty points based on the cards they have left in their hands.
- **Scoring:** After one player has played all his/her cards, the game ends, and that player becomes the winner. The winner is awarded one point for each card that remains in the hands of the other players.

Table 12
Category of actions in DouDiZhu.

Action Category	Description	Num	Example
Pass	Declining to play any cards	\	\
Solo	Any single card	15	6
Pair	Two cards of equal rank	13	66
Triplet	Three cards of equal rank	13	666
Triplet with Solo	A Triplet with a Solo	182	6663
Triplet with Pair	A Triplet with a Pair	156	66 633
Bomb	Four cards of equal rank	13	6666
Bomb with Solos	Bomb with two additional Solo	1183	666 634
Bomb with Pairs	Bomb with two additional Pair	858	66663344
Rocket	Black Joker and Red Joker	1	BR
Chain	Five or more consecutive cards	36	45 678
Pair Chain	Three or more consecutive Pairs	52	445 566
Plane	Two or more consecutive Triplets	45	666 777
Plane with Solos	Plane with each has a distinct Solo	8044	66677734
Plane with Pairs	Plane with each has a distinct Pair	2939	6667773344

The actions in the "Action Category" column highlighted in boldface are variable-length, while the rest are fixed-length.

Table 13
Category of actions in Big2.

Action category	Description	Num	Example
Pass	Declining to play any cards	\	\
Solo	Any single card	52	♢6
Pair	Two cards of equal rank	78	♢6♡6
Triplet	Three cards of equal rank	52	♢6♣6♡6
Straight	Five consecutive cards with different suits	10 200	♠7♡8♢9♣T♡J
Flush	Any five cards of the same suit	5108	♡3♡6♡7♡T♡Q♡A
Full House	Three cards of one rank and two of another rank	3744	♢8♣8♠8♡3♠3
Quads	All four cards of one rank, plus any fifth card	624	♢8♣8♡8♠8♢3
Straight Flush	Five consecutive cards of the same suit	40	♣9♣T♣J♣Q♣K

The actions in the "Action Category" column highlighted in boldface are variable-suit, while the rest are fixed-suit.

Fig. 7. Feature Encoding. All actions and cards in Table 14 are encoded in the first four rows, where the columns indicate the type of card. A '1' in the row signifies presence, while '0' indicates absence. The Identity, camp, number of cards, and one-hot vector in the Table 14 are similar to those encoded in the last four rows.

B.2. State and action space

The state space of Big2 consists of all possible combinations of cards that can be held by the four players, where each player has 13 cards. The total number of possible hands for each player is 10^{28} [45]. Big2 has nine distinct types of actions, such as Pass, Solo, Pair, and others, as indicated by the "Action Category" column in Table 13. The corresponding number of each action category is presented in the "Num" column of the same table. The action space in Big2 consists of 19,899 possible legal actions.

Appendix C. Feature representation

Table 14 describes the feature representation, where action feature, state features, and hidden features are fed into the decision model (refer to Fig. 2(a)). Hidden features are predicted by the opponent model (refer to Fig. 2(b)). This feature representation is similar to DouZero [13] with the following modifications. First, in DouDiZhu, the 54 cards are grouped into 15 ranks, excluding the single Black and Red Joker cards, whereas the remaining cards consist of four each. As a result, Black and Red Jokers are encoded by a 1×2 matrix, while the remaining cards are encoded by a 4×13 matrix. These matrices are flattened and concatenated into a 1×54 matrix, as shown in the first four rows of Fig. 7. Similarly, Big2 has no Black and Red Joker and is composed of 1×52 cards when unfolded. Two additional zeros are added to form a 1×54 matrix. Second, DouDiZhu and Big2 has multiple players. The identity (binary 00 for Landlord, 01 for Peasant-Down, 10 for Peasant-Up, 00 for West, 01 for South, 10 for East, and 11 for North) and camp (binary 1 for the camp of Landlord and 0 for the camp of two Peasants) of these players are encoded into binary values, as shown in last four rows of Fig. 7. This proposed scheme enables a single model that can be applied to all players. Moreover, the features in DouZero are expanded to include four players, enabling their application in both DouDiZhu and Big2 games.

Appendix D. More details of minimum combination search

Minimum combinations are used to estimate the distance to win a game. Eq. (1) is used to compute the minimum combinations denoted as K. This is achieved by considering all possible variable-length/variable-suit actions and treating the remaining cards as minimum decompositions of fixed-length/fixed-suit actions. The minimum combinations of variable-length/variable-suit actions, represented as $Variable(x)$, can be obtained through a deep-first search, while the minimum combinations of fixed-length/fixed-suit actions, denoted by $Fixed[a][b][m][n]$, can be computed using dynamic programming.

Algorithms 2 and 5 illustrates the minimum combination search for Big2 and DouDiZhu, respectively. We preprocessed $Fixed[a][b][y][z]$ by iterating over the variables a, b, y, and z using four loops. If the condition $4a + 3b + 2y + z \leq N$ is satisfied, where N represents the average initial number of cards per player, we invoked the function $Calculate\_Fixed(a, b, y, z)$ to update the array $Fixed[a][b][y][z]$ with the minimum of its current value and the value returned by $Calculate\_Fixed(a, b, y, z)$. Through preprocessing, the minimum combination values of fixed-length/fixed-suit actions are stored in memory so that they can be queried directly by the variable-length/variable-suit search function $Variable(x)$, without the need for repeated computations. This enhances the efficiency of the search process.

Algorithm 5 Minimum Combination Search for DouDiZhu.

Input: The quantity of 15 different rank cards: $x[0...14]$
Output: The minimum combinations K for x
procedure $Variable(x)$

1. Iterate over the elements in x with three for-loops, to find all possible variable-length actions:
   - 1.1 If any **Chain** or **Pair Chain** $\in x$, create a copy of x, denoted as x', and remove the same number of identical elements from x' as this **Chain** or **Pair Chain**. Then call $Variable(x')$ recursively to obtain the minimum value $K = min(K, Variable(x') + 1)$ until the end of the iteration.
   - 1.2 If any **Plane** $\in x$:
     - 1.2.1 Create a copy of x, denoted as x', and remove the same number of identical elements from x' as this **Plane**. Then, call $Variable(x')$ recursively to obtain the minimum value $K = min(K, Variable(x') + 1)$ until the end of the iteration.
     - 1.2.2 If any **Plane with Solos** or **Plane with Pairs** $\in x$: Create a copy of x, denoted as x', and remove the same number of identical elements in x' as this **Plane**, then iterate over x' recursively until the exact number of **Solos** or **Pairs** as **Triplet** in this **Plane** are find, then call $Variable(x')$ recursively to obtain the minimum value $K = min(K, Variable(x') + 1)$ until the end of the iteration.
2. Iterate over x to count the remaining number of **Bomb**, **Triplet**, **Pair**, and **Solo** denoted as a, b, y, and z, respectively. Then, return one of the minimum value $min(K, Fixed[a][b][y][z])$.

end procedure
procedure $Calculate\_Fixed(a, b, y, z)$

$Fixed[a][b][y][z] \leftarrow \min($
$Fixed[a][b][y][z]$, ▷ Current minimum value
$Fixed[a][b][y][z-1]+1$, ▷ Play a Solo
$Fixed[a][b][y-1][z]+1$, ▷ Play a Pair
$Fixed[a][b][y-1][z+2]$, ▷ Split a Pair to two Solos
$Fixed[a][b-1][y][z]+1$, ▷ Play a Triplet
$Fixed[a][b-1][y+1][z+1]$, ▷ Split a Triplet to a Pair and a Solo
$Fixed[a][b-1][y][z+3]$, ▷ Split a Triplet to three Solos
$Fixed[a][b-1][y][z-1]+1$, ▷ Play a Triplet with Solo
$Fixed[a][b-1][y-1][z]+1$, ▷ Play a Triplet with Pair
$Fixed[a-1][b][y][z]+1$, ▷ Play a Bomb
$Fixed[a-1][b+1][y][z+1]$, ▷ Split a Bomb to a Triplet and a Solo
$Fixed[a-1][b][y+2][z]$, ▷ Split a Bomb to two Pairs
$Fixed[a-1][b][y+1][z+2]$, ▷ Split a Bomb to a Pair and two Solos
$Fixed[a-1][b][y][z+4]$, ▷ Split a Bomb to four Solos
$Fixed[a-1][b][y][z-2]+1$, ▷ Play a Bomb with Solos
$Fixed[a-1][b][y-2][z]+1)$ ▷ Play a Bomb with Pairs

return $Fixed[a][b][y][z]$
end procedure
procedure MAIN

1. $Fixed[a][b][y][z]$ is preprocessed beforehand, then it can be directly queried in the $Variable(x)$ function.
2. $K \leftarrow Variable(x)$
3. If **Rocket** $\in x$: Create a copy of x, denoted as x', remove **Rocket** from x', then return $min(K, Variable(x') + 1)$

end procedure

Table 14
Feature representation.

Action feature	Size
The action to be taken	54
State features	
The self-hand cards	54
The union of the other players' hand cards	54
Identity (two bits) repeats 9 times:1 × 18; Camp (one bit) repeats 16 times:1 × 16; The number of left cards: 1 × 20 one-hot vector. These three fields identify the player who will take the action	54
the most recent action	54
The identity, camp, and the number of cards left of the player who takes the most recent action	54
The played cards of the 1st player	54
The played cards of the 2nd player	54
The played cards of the 3rd player	54
The played cards of the 4th player. The played cards for DouDiZhu are filled with zeros.	54
The number of cards left for each player in DouDiZhu: 1 × 20 one-hot vector for Landlord; 1 × 17 one-hot vector for Peasant-Down; 1 × 17 one-hot vector for Peasant-Up. Or the number of cards left for each player in Big2: 1 × 13 one-hot vector for each player (4 players). The last two bits in Big2 are filled with zeros	54
Last 12 historical actions of DouDiZhu or the last 16 historical actions of Big2. The action of the fourth player in DouDiZhu are filled with zeros	4 × 216
hidden features predicted by the opponent model	
Hand cards:4 players × 54. Big2: 4 players; DouDiZhu: 3 players. The hand cards of the fourth player in DouDiZhu are filled with zeros	216
Minimum combinations:4 players × 20. The minimum combinations of the fourth player in DouDiZhu are filled with zeros	80

References

[1] D. Mnih, V. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, 2013, http://dx.doi.org/10.48550/arXiv.1312.5602, arXiv:1312.5602.

[2] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, second edition, MIT Press, 2018.

[3] Y. Song, P.N. Suganthan, W. Pedrycz, J. Ou, Y. He, Y. Chen, Y. Wu, Ensemble reinforcement learning: A survey, Appl. Soft Comput. 149 (2023) 110975, http://dx.doi.org/10.1016/j.asoc.2023.110975.

[4] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489, http://dx.doi.org/10.1038/nature16961.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Mastering the game of go without human knowledge, Nature 550 (7676) (2017) 354–359, http://dx.doi.org/10.1038/nature24270.

[6] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, Science 362 (6419) (2018) 1140–1144, http://dx.doi.org/10.1126/science.aar6404.

[7] J. Perolat, B. De Vylder, D. Hennes, E. Tarassov, F. Strub, V. de Boer, P. Muller, J.T. Connor, N. Burch, T. Anthony, Mastering the game of stratego with model-free multiagent reinforcement learning, Science 378 (6623) (2022) 990–996, http://dx.doi.org/10.1126/science.add4679.

[8] O. Vinyals, I. Babuschkin, W.M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D.H. Choi, R. Powell, T. Ewalds, P. Georgiev, Grandmaster level in StarCraft II using multi-agent reinforcement learning, Nature 575 (7782) (2019) 350–354, http://dx.doi.org/10.1038/s41586-019-1724-z.

[9] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, Dota 2 with large scale deep reinforcement learning, 2019, http://dx.doi.org/10.48550/arXiv.1912.06680, arXiv:1912.06680.

[10] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo, Mastering complex control in MOBA games with deep reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, (04) 2020, pp. 6672–6679, http://dx.doi.org/10.1609/aaai.v34i04.6144.

[11] Y. Wang, A. Bracciali, G. Yang, T. Li, X. Yu, Adversarial behaviours in mixing coins under incomplete information, Appl. Soft Comput. 96 (2020) 106605, http://dx.doi.org/10.1016/j.asoc.2020.106605.

[12] Y. Lu, Y. Zhao, W. Zhou, H. Li, Danzero: Mastering guandan game with reinforcement learning, in: IEEE Conference on Games (CoG), 2023, pp. 1–8, http://dx.doi.org/10.1109/CoG57401.2023.10333179.

[13] D. Zha, J. Xie, W. Ma, S. Zhang, X. Lian, X. Hu, J. Liu, DouZero: Mastering doudizhu with self-play deep reinforcement learning, in: International Conference on Machine Learning, 2021, pp. 12333–12344, http://proceedings.mlr.press/v139/zha21a.html.

[14] E. Zhao, R. Yan, J. Li, K. Li, J. Xing, AlphaHoldem: High-performance artificial intelligence for heads-up no-limit poker via end-to-end reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 4689–4697, http://dx.doi.org/10.1609/aaai.v36i4.20394.

[15] H. Xu, K. Li, H. Fu, Q. Fu, J. Xing, AutoCFR: Learning to design counterfactual regret minimization algorithms, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 5244–5251, http://dx.doi.org/10.1609/aaai.v36i5.20460.

[16] P. Liu, J. Zhou, J. Lv, Exploring the first-move balance point of go-moku based on reinforcement learning and Monte Carlo tree search, Knowl.-Based Syst. 261 (2023) 110207, http://dx.doi.org/10.1016/j.knosys.2022.110207.

[17] D. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A survey of monte carlo tree search methods, IEEE Trans. Comput. Intell. AI Games 4 (1) (2012) 1–43, http://dx.doi.org/10.1109/TCIAIG.2012.2186810.

[18] M. Świechowski, K. Godlewski, B. Sawicki, J. Mańdziuk, Monte Carlo tree search: A review of recent modifications and applications, Artif. Intell. Rev. 56 (3) (2023) 2497–2562, http://dx.doi.org/10.1007/s10462-022-10228-y.

[19] C. He, A review of the application of artificial intelligence in imperfect information games represented by DouDiZhu, in: International Conference on Science and Technology Ethics and Human Future, 2022, pp. 160–166, http://dx.doi.org/10.2991/assehr.k.220701.033.

[20] Y. Zhao, J. Zhao, X. Hu, W. Zhou, H. Li, Full DouZero+: Improving DouDizhu AI by opponent modeling, coach-guided training and bidding learning, IEEE Trans. Games (2023) http://dx.doi.org/10.1109/TG.2023.3299612.

[21] H. Li, Z. Guo, Y. Liu, X. Wang, S. Qi, J. Zhang, J. Xiao, Kdb-D2CFR: Solving multiplayer imperfect-information games with knowledge distillation-based DeepCFR, Knowl.-Based Syst. 272 (2023) 110567, http://dx.doi.org/10.1016/j.knosys.2023.110567.

[22] L.-W. Chen, Y.-R. Lu, Challenging artificial intelligence with multiopponent and multimovement prediction for the card game Big2, IEEE Access 10 (2022) 40661–40676, http://dx.doi.org/10.1109/ACCESS.2022.3166932.

[23] P.I. Cowling, E.J. Powley, D. Whitehouse, Information set monte carlo tree search, IEEE Trans. Comput. Intell. AI Games 4 (2) (2012) 120–143, http://dx.doi.org/10.1109/TCIAIG.2012.2200894.

[24] D. Whitehouse, E.J. Powley, P.I. Cowling, Determinization and information set monte carlo tree search for the card game dou di zhu, in: IEEE Conference on Computational Intelligence and Games, 2011, pp. 87–94, http://dx.doi.org/10.1109/CIG.2011.6031993.

[25] H. Ihara, S. Imai, S. Oyama, M. Kurihara, Implementation and evaluation of information set Monte Carlo tree search for Pokémon, IEEE Int. Conf. Syst., Man, Cybern. (2018) 2182–2187, http://dx.doi.org/10.1109/SMC.2018.00375.

[26] B.K. Demirdöver, Ö. Baykal, F. Alpaslan, Learning to play an imperfect information card game using reinforcement learning, Turk. J. Electr. Eng. Comput. Sci. 30 (6) (2022) 2303–2318, http://dx.doi.org/10.55730/1300-0632.3940.

[27] N. Zolboot, Q. Johnson, D. Shen, A. Redei, Hearthstone battleground: An AI assistant with Monte Carlo tree search, in: Proceedings of 37th International Confer, vol. 82, 2022, pp. 131–140, http://dx.doi.org/10.29007/mn6n.

[28] S. Nashed, S. Zilberstein, A survey of opponent modeling in adversarial domains, J. Artificial Intelligence Res. 73 (2022) 277–327, http://dx.doi.org/10.1613/jair.1.12889.

[29] S. Ganzfried, K.A. Wang, M. Chiswick, Bayesian opponent modeling in multiplayer imperfect-information games, 2022, http://dx.doi.org/10.48550/arXiv.2212.06027, arXiv:2212.06027.

[30] N. Mizukami, Y. Tsuruoka, Building a computer mahjong player based on Monte Carlo simulation and opponent models, in: IEEE Conference on Computational Intelligence and Games, 2015, pp. 275–283, http://dx.doi.org/10.1109/CIG.2015.7317929.

[31] Z. Tang, Y. Zhu, D. Zhao, S.M. Lucas, Enhanced rolling horizon evolution algorithm with opponent model learning, IEEE Trans. Games (2020) http://dx.doi.org/10.1109/TG.2020.3022698.

[32] A. Kai Qin, Vicky Ling Huang, Ponnuthurai N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol. Comput. 13 (2) (2009) 398–417, http://dx.doi.org/10.1109/TEVC.2008.927706.

[33] H. Xuan, W. Li, B. Li, An artificial immune differential evolution algorithm for scheduling a distributed heterogeneous flexible flowshop, Appl. Soft Comput. 145 (2023) 110563, http://dx.doi.org/10.1016/j.asoc.2023.110563.
[34] O. Chapelle, B. Scholkopf, A. Zien, Semi-supervised learning (chapelle, o. et al., eds.; 2006), IEEE Trans. Neural Netw. 20 (3) (2009) 542, http://dx.doi.org/10.1109/TNN.2009.2015974.
[35] S.A. Hosseini, A. Abbaszadeh Shahri, R. Asheghi, Prediction of bedload transport rate using a block combined network structure, Hydrol. Sci. J. 67 (1) (2022) 117–128, http://dx.doi.org/10.1080/02626667.2021.2003367.
[36] K. Adane, B. Beyene, Machine learning and deep learning based phishing websites detection: The current gaps and next directions, Rev. Comput. Eng. Res. 9 (1) (2022) 13–29, http://dx.doi.org/10.18488/76.v9i1.2983.
[37] J. Li, S. Koyamada, Q. Ye, G. Liu, C. Wang, R. Yang, L. Zhao, T. Qin, T.-Y. Liu, H.-W. Hon, Suphx: mastering mahjong with deep reinforcement learning, 2020, http://dx.doi.org/10.48550/arXiv.2003.13590, arXiv:2003.13590.
[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, http://dx.doi.org/10.48550/arXiv.1707.06347, arXiv:1707.06347.
[39] H. Charlesworth, Application of self-play reinforcement learning to a four-player game of imperfect information, 2018, http://dx.doi.org/10.48550/arXiv.1808.10442, arXiv:1808.10442.
[40] Y. Guan, M. Liu, W. Hong, W. Zhang, F. Fang, G. Zeng, Y. Lin, Perfectdou: Dominating doudizhu with perfect information distillation, Adv. Neural Inf. Process. Syst. 35 (2022) 34954–34965, http://dx.doi.org/10.48550/arXiv.2203.16406.
[41] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533, http://dx.doi.org/10.1038/nature14236.
[42] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, 2016, pp. 1928–1937, https://proceedings.mlr.press/v48/mniha16.html.
[43] Y. You, L. Li, B. Guo, W. Wang, C. Lu, Combinatorial q-learning for dou di zhu, in: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 16, (1) 2020, pp. 301–307, http://dx.doi.org/10.1609/aiide.v16i1.7445.
[44] Q. Jiang, K. Li, B. Du, H. Chen, H. Fang, Deltadou: Expert-level doudizhu AI through self-play, in: International Joint Conferences on Artificial Intelligence, 2019, pp. 1265–1271, http://dx.doi.org/10.24963/ijcai.2019/176.
[45] H.C.J. Chow, H.C. Cheng, Deep reinforcement learning for big two, 2022, https://github.com/johnnyhoichuen/big2-rl/blob/main/DRL%20with%20Big2.pdf.
[46] D. Zha, K.-H. Lai, Y. Cao, S. Huang, R. Wei, J. Guo, X. Hu, Rlcard: A toolkit for reinforcement learning in card games, 2019, http://dx.doi.org/10.48550/arXiv.1910.04376, arXiv:1910.04376.
[47] G. Fernando Sugiyanto, W.K. Tai, A rule-based AI method for an agent playing big two, Appl. Sci. 11 (9) (2021) 4206, http://dx.doi.org/10.3390/app11094206.
[48] A. Abbaszadeh Shahri, C. Shan, S. Larsson, A novel approach to uncertainty quantification in groundwater table modeling by automated predictive deep learning, Natural Resour. Res. 31 (3) (2022) 1351–1373, http://dx.doi.org/10.1007/s11053-022-10051-w.
[49] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, Y. Wang, Transformer in transformer, Adv. Neural Inf. Process. Syst. 34 (2021) 15908–15919, http://dx.doi.org/10.48550/arXiv.2103.00112.
[50] C.-K. Yeh, C.-Y. Hsieh, H.-T. Lin, Automatic bridge bidding using deep reinforcement learning, IEEE Trans. Games 10 (4) (2018) 365–377, http://dx.doi.org/10.1109/TG.2018.2866036.
[51] N. Mignoni, R. Carli, M. Dotoli, Optimal decision strategies for the generalized cuckoo card game, IEEE Trans. Games (2023) http://dx.doi.org/10.1109/TG.2023.3239795.
[52] P. Zhang, A novel feature selection method based on global sensitivity analysis with application in machine learning-based prediction model, Appl. Soft Comput. 85 (2019) 105859, http://dx.doi.org/10.1016/j.asoc.2019.105859.