Comparing Tree Search and Reinforcement Learning Approaches for King and Courtesan Game

Anonymous Author
École Polytechnique
CSC_52081_EP Advanced Machine Learning and Autonomous Agents

***Abstract*—This project presents a comparative analysis of traditional tree search algorithms and modern reinforcement learning techniques to solve the board game King and Courtesan. We implement and evaluate three distinct approaches: (1) Iterative Deepening Alpha-Beta pruning with move ordering, (2) Deep Q-Networks (DQN) with infrequent weight updates, and (3) Monte Carlo Policy Gradient (REINFORCE). To address the challenges of learning in this adversarial environment with sparse rewards, we also employ gradient-free optimization methods, specifically Cross-Entropy Method (CEM) and Self-Adapting Evolution Strategies, to train our neural network agents. Our implementation bridges a robust Java-based game environment with Python-based Reinforcement Learning agents through a socket communication interface. Experimental results reveal the relative strengths of each approach: tree search excels with sufficient computational resources but scales poorly with time constraints, while properly optimized neural network agents demonstrate faster decision-making with competitive performance. Notably, gradient-free optimization outperforms traditional gradient-based methods for training game-playing neural networks in this specific environment. This work contributes to the understanding of algorithm selection for adversarial board games and demonstrates the potential of hybrid optimization techniques that combine the representational power of deep networks with the robustness of evolutionary methods.**

I. Introduction

In this work, we investigate the application of modern Reinforcement Learning techniques to solve the two-player board game "King And Courtesan," comparing them against traditional tree search algorithms. Specifically, we examine:

1) The performance of Iterative Deepening Alpha-Beta (IDAB) pruning with move ordering as a baseline approach
2) Deep Q-Networks (DQN) with infrequent weight updates as a value-based deep RL method
3) Monte Carlo Policy Gradient (REINFORCE) as a policy-based deep RL method
4) Gradient-free optimization techniques (CEM and ES) for learning neural network weights

Game-playing AI presents unique challenges, particularly for complex adversarial board games. While conventional tree search algorithms like Alpha-Beta have demonstrated robust performance, they often struggle with games featuring high branching factors or when computational resources are limited. Reinforcement Learning offers a promising alternative by learning effective policies through experience rather than explicit search.

Our approach builds on previous work comparing search and learning methods for classic games like Chess and Go, but with several key innovations. First, we apply these methods to the relatively new game of King and Courtesan, which features distinct mechanics with interesting challenges. Second, we implement a cross-language framework that bridges our Java-based game environment with Python-based Deep Learning agents. Third, we compare different optimization techniques, namely gradient-based and gradient-free approaches.

Our experiments demonstrate that while tree search methods excel with sufficient computational resources, they scale poorly with time constraints. In contrast, neural network agents trained with gradient-free optimization show competitive performance with constant decision time.

We found that gradient-free optimization methods outperform traditional gradient-based approaches for training game-playing neural networks in this environment. This suggests important implications for similar adversarial environments with sparse rewards.

Despite these promising results, we identify some limitations including computational budget constraints, communication overhead in our cross-language implementation, and potential blind spots in self-play training.

The code is available here.

II. Background

A. The King and Courtesan Game

King and Courtesan is a drawless two-player army game invented by Mark Steere in 2022. You can play it online here. The game is played on a 6×6 grid where each player controls a king and multiple courtesans, with the objective of either moving their king to the opponent's home square or capturing the opponent's king.

Players alternate turns, with each player making exactly one move per turn from three possible types:

1) **Non-capturing moves:** Kings and courtesans can move to adjacent unoccupied squares in any of the three forward directions.
2) **Capturing moves:** Kings and courtesans can move to adjacent enemy-occupied squares in any of eight directions, capturing the enemy piece by replacement.
3) **Exchange moves:** A player can exchange its king's position with a friendly courtesan located in any of the three forward directions.

The game terminates when either a player captures the opponent's king or a player successfully moves their king to the opponent king's starting position. This guaranteed termination condition eliminates the possibility of draws, making King and Courtesan particularly suitable for AI research as it always leads to a definitive outcome.

The game's branching factor varies considerably depending on the board state. Theoretical analysis suggests a maximum branching factor of approximately 120 moves, though empirical observation shows the typical branching factor in play is around 22-27 moves.

B. *Traditional Search Algorithms*

Iterative Deepening Alpha-Beta (IDAB) pruning is a classic game tree search algorithm that combines the completeness of breadth-first search with the memory efficiency of depth-first search. The algorithm explores the game tree layer by layer, increasing the maximum search depth with each iteration until finding a winning sequence or exhausting the allocated time limit.

The search alternates between two recursive functions:

- maxMin(s, d, α, β): Selects the move that maximizes the minimum value the opponent can force
- minMax(s, d, α, β): Selects the move that minimizes the maximum value the adversary can achieve

Both functions maintain α and β bounds to prune branches that cannot influence the final decision. A critical enhancement is move ordering, which significantly improves pruning efficiency by examining promising moves first.

C. *Reinforcement Learning Approaches*

For complex games with large state spaces, tabular reinforcement learning methods like TD Learning, QLearrning and SARSA are impractical. Instead, we focus on Deep Reinforcement Learning approaches:

*1) **Deep Q-Networks (DQN):*** DQN extends the Q-learning algorithm with neural network function approximation. The network is trained to minimize the loss:

$$L(\theta) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2] \quad (1)$$

Where θ^- represents parameters of a target network that is updated less frequently than the online network. Key components include experience replay to break correlations between consecutive samples and target networks with infrequent updates to stabilize learning.

*2) **Monte Carlo Policy Gradient (REINFORCE):*** REINFORCE directly optimizes a parameterized policy without requiring value function estimation. Following the policy gradient theorem, parameters are updated in the direction:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t] \quad (2)$$

Where G_t is the return from time step t, and $\nabla_\theta \log \pi_\theta(a|s)$ is the score function.

*3) **Gradient-Free Optimization:*** While gradient-based methods are standard for training neural networks, they face challenges in game-playing domains with sparse rewards and non-stationary environments. Gradient-free alternatives include:

- **Cross-Entropy Method (CEM):** A population-based evolutionary strategy that maintains and refines a probability distribution over parameters
- **(1+1)-SA-ES:** A single-individual evolution strategy with self-adapting step sizes for efficient local parameter space exploration

These methods directly search the parameter space to find effective policies without computing gradients, potentially offering greater robustness in challenging domains.

III. Methodology/Approach

A. *Implementation Architecture*

Our implementation consists of three main components: a core game environment and our Iterative Deepening Alpha Beta with move ordering baseline, both written in Java, and a Python wrapper for Reinforcement Learning agent development. To bridge these components, we developed a cross-language framework using socket-based communication.

The Java environment implements the complete game logic and rules enforcement. We model the board as a 6×6 matrix where each cell contains one of five possible states: RED_KING, BLUE_KING, RED_COURTESAN, BLUE_COURTESAN, or EMPTY.

Our Python wrapper implements the standard OpenAI Gym interface, providing methods such as `reset()`, `step(action)`, and `render()`. When an agent selects an action, the wrapper:

1) Encodes the action into a standardized format
2) Sends it to the Java server via socket connection
3) Receives the updated game state, reward signal, termination and truncation flags
4) Converts this information back into tensor representations suitable for neural network processing

This architecture combines the performance and correctness guarantees of our Java implementation with the flexibility of Python-based machine learning libraries.

B. *State and Action Representation*

For state representation, we encode the board as a tensor with 6 channels, each representing a 6×6 grid:

1) Red king position (1 where present, 0 elsewhere)
2) Red courtesans positions
3) Blue king position
4) Blue courtesans positions
5) Empty squares
6) Current player (filled with 1s for current player, 0s for opponent)

For the action space, we implement a fixed-size output with action masking:

1) The network outputs a fixed-size vector representing all possible "from-to" position pairs ($36 \times 36 = 1296$ outputs)
2) During action selection, we apply a mask that assigns negative infinity to illegal moves
3) The selection process then considers only legal moves when determining the maximum Q-value or sampling from policy probabilities

C. *Iterative Deepening Alpha Beta Agent*

Our IDAB implementation explores the game tree through increasing depth limits, with several key components:

- A sophisticated heuristic function incorporating multiple strategic considerations:
  - Difference in number of pieces players betwen players
  - Mobility difference (available legal moves)
  - Potential piece captures
  - King's distance to the opponent's base
  - Opponent king's distance to current player's base
  - Average distance between pieces and the opponent's king
  - King safety factors
  - Terminal state detection (win/loss conditions)
- Move ordering based on heuristic evaluations from previous iterations
- Alpha-Beta pruning with time-constrained search
- Extreme values for terminal states (Integer.MAX_VALUE for wins, Integer.MIN_VALUE for losses)

D. *DQN v2015 Agent*

Our DQN implementation follows the architecture and techniques introduced in the improved 2015 version:

- **Neural Network Architecture:**
  - Input layer: 6 channels $\times$ 6×6 board representation
  - Three convolutional layers with 64, 128, and 128 filters (3×3 kernel, stride 1)
  - Two fully connected layers of 512 and 256 units
  - Output layer with 1296 units
  - ReLU activation functions throughout
- **Experience Replay:** Buffer capacity of 100,000 with mini-batches of 32
- **Target Network:** Updated every 1000 training steps
- **Action Selection:** ϵ-greedy policy with annealing from 1.0 to 0.1
- **Reward Structure:**
  - +1 for winning
  - -1 for losing
  - 0 for intermediate steps
- **Action Masking:** Setting logits of illegal moves to negative infinity

E. *REINFORCE Algorithm*

Our REINFORCE network has a similar architecture:

- **Neural Network Architecture:**
  - Input layer: 6 channels $\times$ 6×6 board representation
  - Two convolutional layers with 64 and 128 filters
  - Two fully connected layers of 512 and 256 units
  - Output layer with 1296 units (softmax activation)
- **Same reward structure**
- **Action Masking**

F. *Gradient-Free Optimization*

We employ two gradient-free optimization methods:

1) *Cross-Entropy Method (CEM):*

- **Algorithm:** Maintain and refine a probability distribution over parameters
  1) Initialize distribution parameters μ_0 and Σ_0
  2) For each iteration:
     - Sample population $\{\theta_1, \theta_2, ..., \theta_n\}$ from distribution
     - Evaluate fitness $J(\theta_i)$ for each individual
     - Select top k elite individuals
     - Update distribution parameters based on elite population
- **Implementation Details:**
  - Population size: 100 individuals
  - Elite fraction: 20%
  - Minimum variance: $\sigma^2_{min} = 0.01$
  - One mean and one variance parameters per dimension of θ
  - Parallelized evaluation across multiple CPU cores

2) *(1+1)-SA-ES:*

- **Algorithm:** Maintain a single solution with adaptive step size.
  Here τ is a self-adaptation learning rate
  1) Initialize parameters θ_0 and step size σ_0
  2) For each iteration:
     - Sample perturbation $\epsilon_\sigma \sim N(0, I)$
     - Create offspring $\sigma' = \sigma_t e^{\tau \epsilon_\sigma}$
     - Sample perturbation $\epsilon_\theta \sim N(0, I)$
     - Create offspring $\theta' = \theta_t + \sigma_t \cdot \epsilon_\theta$
     - Evaluate parent and offspring
     - Accept offspring if superior, adjusting step size accordingly
- **Implementation Details:**
  - Initial step size $\sigma_0 = 0.1$

G. *Experience Collection Strategy*

We implement a self-play framework where agents play against themselves or previous versions. This approach:

- Eliminates the need for a separate opponent agent
- Creates a natural curriculum where agent capability increases over time

- Avoids potential bias from playing against any single fixed strategy

For each algorithm, we use a player perspective approach where states are encoded from the current player's viewpoint, ensuring consistent policy learning regardless of playing as Red or Blue.

IV. Results and Discussion

A. Evaluation Metrics

We evaluate agent performance using the following metrics:

- Win rate: Percentage of games won against specific opponents
- Average game length: Number of moves until termination
- Computation time: Processing time required for decision-making
- Adaptability: Performance against different opponent types

B. Training Convergence

Figure 1 shows the training progression for both DQN and REINFORCE agents, plotting expected returns against training iterations. Notable observations include:

- DQN exhibits higher initial variance but converges faster in early training
- REINFORCE displays more stable progression with lower variance
- Gradient-free optimization shows different convergence patterns than gradient-based methods

Fig. 1. Training convergence for DQN, REINFORCE and gradient-free optimization methods. The x-axis shows number of training episodes, and the y-axis shows expected returns.

C. Head-to-Head Performance

Table 1 presents the outcomes of 100 games played between each pair of agents, with entries showing the win rate of the row agent against the column agent:

TABLE I
WIN RATES IN HEAD-TO-HEAD MATCHUPS BETWEEN DIFFERENT AGENTS. EACH CELL REPRESENTS THE WIN PERCENTAGE OF THE ROW AGENT AGAINST THE COLUMN AGENT.

Agent	IDAB	DQN	REINFORCE
IDAB	-	30%	35%
DQN	70%	-	55%
REINFORCE	65%	45%	-

	CEM	(1+1)-SA-ES
IDAB	25%	35%
DQN	40%	47%
REINFORCE	34%	38%

Key findings from head-to-head competition:

- IDAB is the least performant
- DQN outperforms REINFORCE in direct competition
- The first player advantage is evident across all matchups
- Higher computational budgets significantly improve IDAB performance

D. Decision Time Analysis

Figure 2 compares the computation time required by each agent to make decisions throughout the game:

Fig. 2. Comparison of decision time (in milliseconds) required by each agent type across different board positions. IDAB shows increasing computation time in complex mid-game positions, while neural network approaches maintain near-constant decision times.

- IDAB shows variable decision times, increasing significantly in complex mid-game positions
- Both neural network approaches maintain near-constant decision times regardless of position complexity
- IDAB with limited time performs significantly worse than with extended thinking time

E. Ablation Studies

To understand the contribution of different components to agent performance, we conducted ablation studies:

- DQN and REINFORCE without action masking failed to learn effectively
- Gradient-free optimization outperformed gradient-based methods for both network architectures
- Heuristic weights in IDAB significantly impact strategic tendencies

V. Conclusions

This study compared traditional tree search techniques with modern Reinforcement Learning approaches for solving the King and Courtesan board game. Our investigation yields insights into the relative strengths and weaknesses of these paradigms.

The Iterative Deepening Alpha-Beta algorithm with move ordering demonstrates robust performance and explainable decision-making, making it an excellent baseline approach. Its primary limitations are computational demands and dependence on hand-crafted heuristics. Deep Q-Networks performs really well in learning complex positional patterns without explicit search, while REINFORCE exhibits strong performance with simplified implementation.

We found that gradient-free optimization methods (CEM and SA-ES) provide substantial benefits for training neural networks in this domain, outperforming traditional gradient-based approaches. This suggests that these methods may be more suitable for games with sparse rewards and complex action spaces.

Some limitations deserve mention. First, our computational budget constrained the depth of IDAB search and the training time for neural networks. Second, our implementation architecture introduces overhead in the Java-Python communication layer. Third, the self-play training paradigm may create blind spots against novel strategies.

Future work could explore promising directions:

1) Hybrid approaches combining tree search with learned evaluation functions
2) Attention-based architectures to better capture long-range piece interactions

3) Training the networks against a fixed base line policy

By comparing traditional game-playing AI techniques with modern Reinforcement Learning approaches, this work contributes to our understanding of both paradigms' strengths and limitations, potentially informing algorithm selection for similar environments.

REFERENCES

[1] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293-326, 1975.

[2] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529-533, 2015.

[3] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229-256, 1992.

[4] R. Y. Rubinstein and D. P. Kroese, "The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning," *Springer*, 2004.

[5] D. Silver et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354-359, 2017.

[6] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[7] M. Steere, "King and Courtesan," May 2022. [Online]. Available here