Truf State-Space Evaluation for Expected Value Optimization Under Partial Information Utilizing Dynamic Programming, Divide and Conquer, and Greedy Heuristics

Surya Suharna - 18223075
Information Systems and Technology Study Program
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: surya.suharna@gmail.com , 18223075@std.stei.itb.ac.id

***Abstract*—**Truf is a trick-taking card game governed by partial information and complex tactical decision-making. Facing a massive state space prone to an intractable combinatorial explosion, a naive search algorithm yields an astronomical worst-case complexity driven by exponential and factorial growth, making real-time computation fundamentally unviable. This paper presents a comprehensive optimization framework combining a Divide and Conquer paradigm via Perfect Information Monte Carlo with a Dynamic Programming engine. By decomposing the imperfect information state into fully determinized scenarios, the DP algorithm recursively solves independent sub-problems to maximize expected utility. To achieve operational efficiency, three strategic layers are seamlessly integrated into a continuous pipeline: memoization via transposition tables to eliminate redundant calculations from convergent sequences, alpha-beta pruning to aggressively compress the depth-first adversarial search tree, and a localized greedy heuristic for move ordering to accelerate early pruning cutoffs. The analysis demonstrates that this hybrid architecture drastically minimizes the active search space and mitigates the severe computational complexity. Consequently, this approach effectively transforms an otherwise prohibitive computational hurdle into a highly efficient, deployable system for real-time strategic decision-making in the Truf game.

Keywords*—Dynamic Programming; Truf Game; Partial Information; Monte Carlo; Divide and Conquer; Greedy Heuristics; Computational Complexity.*

I. Introduction

The Truf card game is a popular traditional game in Indonesia. Mechanically, Truf is classified as a trick-taking game, where players take turns playing one card in each round (a "trick"), and the player with the highest-valued card according to the suit and trump rules wins the round.

The main complexity in Truf is not merely the number of card combinations, but rather the uncertainty of information (partial information). During the game, a player does not know the exact cards held by the opponents. Players must make optimal moves based solely on the cards in their hand, the cards that have been played on the table, and the probability of the remaining cards.

In the realm of Algorithm Strategies, sequential decision-making of this nature can be solved using the Dynamic Programming (DP) approach. However, due to the element of uncertainty, purely deterministic DP cannot be used directly. This paper will model how to combine Monte Carlo simulations to determine the game state, and then apply Dynamic Programming to each simulation to find the move that maximizes the expected value of winning tricks.

Mathematically, evaluating such an adversarial search space across randomized environments introduces a staggering computational hurdle. In a naive, unoptimized Minimax configuration across W simulated worlds with N remaining cards, the worst-case time complexity escalates exponentially due to the simultaneous permutations of card choices available to all four players.

$$Naive\ Complexity = O(W.(N!)^4)$$

For a standard full game where $N = 13$, this combinatorial explosion yields an astronomical state space exceeding 1.5×10^{39} operations per world, a threshold that is fundamentally intractable for real-time systems.

Consequently, it can be concluded that the computational burden generated by this naive approach is profoundly massive and computationally prohibitive. This staggering threshold exceeding 10^{39} operations underscores that achieving real-time decision-making in the Truf card game is completely intractable without aggressive optimization strategies designed to drastically prune and compress the search space.

II. Theoretical Basis

A. Truf Game as a Partial Information Game

Mathematically, the Truf game can be modeled as a sequential decision-making process with partial information. On each turn, a player is in a state S. his state consists of deterministic information and hidden information, which can be formulated as:

$$S = (H, P, U)$$

Where H is the set of cards in the player's hand. P is the public knowledge, including the history of cards played on the

table and previous tricks and U is the set of unknown cards held by the opponents.

Because U is uncertain, a player cannot deterministically predict the state transition S' after taking an action (playing a card) $a \in A(S)$, where $A(S)$ is the set of legal cards that can be played in state S [1].

B. *Dynamic Programming*

Dynamic Programming (DP) is a mathematical optimization method based on the principle of optimal substructure, where an optimal solution to a complex problem incorporates optimal solutions to its subproblems [2]. If we temporarily assume the game has perfect information (all cards are known), the game transforms into a deterministic Markov Decision Process (MDP). The general recursive relation for a maximization problem in DP is formulated as:

$$f_n(s) = max_x\{c(s,x) + f_{n-1}(s')\}$$

Where $fn(s)$ is is the optimal value at stage n with state s, x is the decision taken, $c(s,x)$ is the immediate gain, and s' is the resulting state for the next stage [1].

In the Truf game system, stages are represented by each round of card play (a trick). The system uses a recursive function to maximize the expected trick count. Assuming perfect information, the Bellman Equation for this game is:

$$V(S) = max_{a \in A(S)}(R(S,a) + V(T(S,a)))$$

Where $A(S)$ is the set of legal cards, $R(S,a)$ is the reward (1 if the trick is won, 0 otherwise), and $T(S,a)$ is the transition to the next game state [2].

C. *Memoization*

Memoization is an optimization technique used in Dynamic Programming to store the results of expensive function calls and return the cached result when the same inputs occur again [1]. It transforms a naive recursive process with exponential complexity into a more efficient one by ensuring that each unique subproblem is solved only once.

Tahap 1:

$$f_1(s) = \min_{x_1}\{c_{sx_1} + f_2(x_1)\}$$

s \ x_1	$f_1(s,x_1) = c_{sx1} + f_2(x_1)$			Solusi Optimum	
	2	3	4	$f_1(s)$	x_1^*
1	13	11	11	11	3 atau 4

Fig 1. Memoization Table Example (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/25-Program-Dinamis-(2026)-Bagian1.pdf)

D. *Divide and Conquer*

Divide and Conquer is a fundamental algorithm design paradigm that resolves a macro-problem by recursively breaking it down into smaller instances [7]. The standard operational pipeline consists of three explicit phases [7]:

1. Divide: Partition the original problem instance into several smaller upa-persoalan (sub-problems) that are similar to the original problem but smaller in scale.
2. Conquer: Solve each sub-problem independently. If the sub-problem size is sufficiently small, solve it directly; otherwise, apply the paradigm recursively.
3. Combine: Merge the individual solutions of the sub-problems to construct the final solution for the original macro-problem.

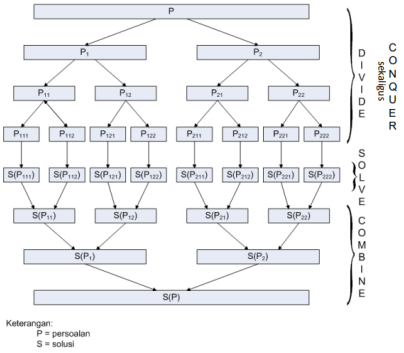

Fig 2. Divide and Conquer Visualization Concept (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/07-Algoritma-Divide-and-Conquer-(2026)-Bagian1.pdf)

E. *Greedy Heuristics*

Greedy algorithms represent one of the most popular methodologies for solving optimization problems, which formally aim to find an optimal solution through either maximization or minimization [8]. The core principle relies on making the best possible choice at any given step without considering future consequences, hoping that a successive chain of locally optimal choices will ultimately converge into a globally optimal solution [8].

To build this solution step-by-step, the algorithm operationally maintains a Candidate Set (C) from which a Solution Set ($S \subseteq C$) is progressively formed. At each decision point, a Selection Function (σ) filters the elements to choose the locally optimal candidate $x^* = argmax_{x \in C} g(x)$ according to a heuristic metric $g(x)$. This candidate is immediately verified by a Feasibility Function (ρ), ensuring that $\rho(SU\{x^*\}) = true$ complies with all structural constraints before insertion. This iterative process repeats until the final compiled solution maximizes or minimizes the global Objective Function ($f(S)$).

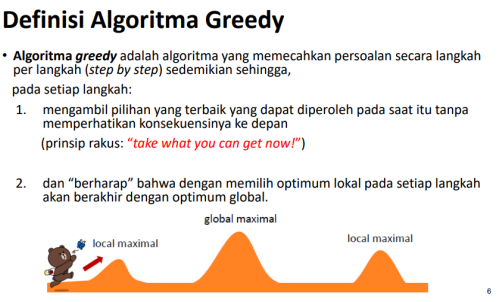

Fig 3. Greedy Algorithm Definitionn (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/04-Algoritma-Greedy-(2026)-Bag1.pdf)

Within the architectural context of tree searches, this selection mechanism is frequently adapted as a Greedy Heuristic. When an algorithm leverages this greedy strategy not to dictate absolute final decisions, but rather to establish a dynamic priority order for evaluation based on the localized ranking metric $g(x)$, the approach becomes exceptionally powerful for accelerating the pruning and elimination of unpromising branches within the search tree [8].

F. Perfect Information Monte Carlo (PIMC)

PIMC is a strategy for solving imperfect information games by using determinization. The algorithm samples S possible deterministic scenarios (worlds) that are consistent with the player's observations. Each world is then solved as a perfect information game, and the results are averaged to estimate the expected value of each action [3].

To address the uncertainty of hidden opponent cards, the algorithmic approach generates N simulations of possible random card distributions. For each simulated scenario w_i, the Dynamic Programming algorithm calculates the optimal evaluation. The expected value $\mathbb{E}$ of playing card a is:

$$\mathbb{E}[V(S,a) \approx \frac{1}{N} \sum_{i=1}^{N} V(w_i, a)$$

The optimal decision $a^* = arg\,max \mathbb{E}[V(S,a)]$.

G. Alpha-Beta Pruning

Alpha-Beta Pruning is a search algorithm that seeks to decrease the number of nodes evaluated by the minimax algorithm in its search tree [4]. It maintains two values, alpha (the minimum score the maximizing player is assured of) and beta (the maximum score the minimizing player is assured of).

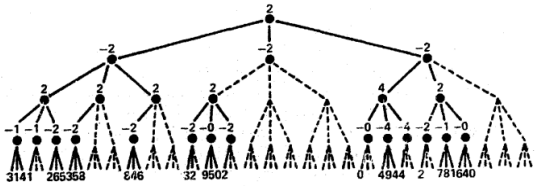

Fig 4. Alpha Beta Prunning Mechanism (Source: D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," Artificial Intelligence, vol. 6, no. 4, pp. 293-326, 1975.)

During the recursive DP evaluation of a card play, if the algorithm finds that a certain move leads to a worse outcome than a previously examined move, it prunes the remaining branches of that scenario. This integration allows the bot to search deeper into the game tree within a limited time frame.

H. Maximizing Expected Value

Rooted in Expected Utility Theory [5], a rational agent facing uncertainty evaluates an action a based on its Expected Utility. The expected utility is calculated as the probability-weighted average of the utilities of all possible outcomes [6]. Let O be the set of possible outcomes, $P(o|a)$ be the conditional probability of outcome o given action a, and $U(o)$ be the utility of that outcome.

$$EU(a) = \sum_{o \epsilon O} P(o|a) U(o)$$

I. Truf Game Rules

The game is played by 4 players, either individually or in pairs. It uses a standard 52-card deck without Jokers, and each player is dealt exactly 13 cards. The main objective is to score points by winning rounds (tricks) according to the initial prediction (bid) made by the player at the start of the round.

The absolute hierarchy of card strength from highest to lowest is A – K – Q – J – 10 – 9 – 8 – 7 – 6 – 5 – 4 – 3 – 2. In each round, one of the four suits (Spades, Hearts, Clubs, or Diamonds) is designated as the "Trump" suit. A Trump card has the highest tier and will always beat cards from any other suit, regardless of their rank.

Before the card play begins, each player must guess how many tricks they believe they can win with their current hand. Usually, this is done simultaneously by placing one card face down on the table. The value of this card represents the player's prediction (e.g., placing a 4 means the player predicts they will win exactly 4 tricks).

The first player plays a card face up. The following players take turns and must follow the leading suit if they have it. If a player does not have the requested suit, they have two options:

1. Discard a random card of another suit (automatically losing the trick).
2. Play a Trump card to "cut" and win the round. Traditionally in Indonesia, a played Trump card must be placed face down. The winner of the trick is the player with the highest card of the leading suit or the highest Trump card (if any). The winner then leads the next trick.

Scoring is calculated at the end of the round by comparing the actual tricks won with the initial bid. In the strict "Exact Win" rule variant, players are required to win the exact number of tricks they bid. Winning fewer tricks (under-tricking) or more tricks (over-tricking) than the bid results in penalty or minus points.

III. Approach and Methodology

This section details the architectural design and algorithmic pipeline used to develop the intelligent agent for the Truf game. The methodology integrates Perfect Information Monte Carlo (PIMC) with a Dynamic Programming (DP) search engine.

A. System Process Pipeline

The decision-making pipeline starts with the Current Table State & Cards in Hand as input. To handle the uncertainty of hidden cards, the framework implements a Divide and Conquer paradigm through the Perfect Information Monte Carlo (PIMC) module. Instead of confronting the intractable partial-information game tree as a single entity, the algorithm divides the core problem of uncertainty by randomizing the opponents' unknown cards into W independent, fully determinized sub-problems ("worlds").

Each of these conquered worlds represents a perfect-information scenario evaluated by the Dynamic Programming engine, which is accelerated by three tightly integrated optimization layers to guarantee real-time computational efficiency: Memoization caches previously evaluated unique states within a transposition table to bypass redundant calculations; Alpha-Beta Pruning compresses the search space by cutting off suboptimal branches; and Greedy Heuristics employs a greedy move-ordering mechanism to prioritize high-strength cards (such as high-value trumps), thereby dramatically accelerating early alpha-beta cutoffs.

After conquering all independent scenarios, the solutions are recombined by the Expected Value module, which aggregates the average projected scores for each legal move to confidently output the final Optimal Decision.

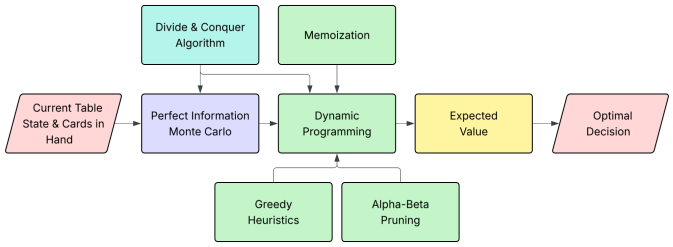

Fig 5. Flow System Process (Source: Author's Diagram with LucidApp)

This snippet demonstrates the integration of PIMC simulations to calculate the highest expected value alongside a recursive Dynamic Programming search optimized by memoization and alpha-beta pruning.

```
Function DP(s, α, β, isMax):
    If terminal(s) return score(s)
    If s ∈ Memo return Memo[s]

    // Greedy Heuristics
    ordered_moves = GreedyMoveOrdering(moves(s))

    If isMax:
        v = -∞
        For m ∈ ordered_moves:
            v = max(v, DP(sim(s, m), α, β,
false))
            α = max(α, v)
            If β ≤ α break // Alpha Cutoff
        return Memo[s] = v
    Else:
        v = +∞
        For m ∈ ordered_moves:
            v = min(v, DP(sim(s, m), α, β,
true))
            β = min(β, v)
            If β ≤ α break // Beta Cutoff
        return Memo[s] = v
```

B. State and Action Representation

The game state s at any point within a trick is mathematically defined as a tuple:

$$s = (H_0, H_1, H_2, H_3, K_{table}, L, T, p_{idx})$$

Where H_i set of remaining cards in player i's hand, where $i \in \{0, 1, 2, 3\}$. K_{table} is the ordered list of cards currently played on the table for the active trick. L is the leading suit of the current trick ($L \in \{Spade, Hearts, Clubs, Diamond, \emptyset\}$. T is the designated Trump suit for the entire round. p_{idx} The index of the active player whose turn it is to move.

The action space $A(s)$ represents the set of all legal cards that the active player p_{idx} can play from their current hand H_{pidx}. The generation of $A(s)$ is strictly governed by the following conditional rules based on the leading suit L.

1. Suit-Following Constraint

   If a trick has already been initiated by a previous player ($L \neq \emptyset$) and the active player possesses at least one card matching that suit ($H_{pidx} \cap L \neq \emptyset$), the action space is restricted exclusively to those matching cards:

   $$A(s) = \{c \in H_{pidx} \mid suit(c) = L\}$$

2. Void Condition

   If the active player possesses no cards of the leading suit ($H_{pidx} \cap L \neq \emptyset$) the player is declared "void" in suit L. The suit constraint is lifted, and the action space expands to include the player's entire remaining hand:

   $$A(s) = H_{pidx}$$

   Under this void condition, the system gains the tactical flexibility to either discard an low-value card from another non-trump suit to minimize risk, or play a card from the Trump suit T to "cut" and seize the trick win.

C. PIMC World Generation Mechanism

The PIMC module operationalizes the Game State s and Action Space constraints to construct N deterministic worlds through a systematic pipeline. First, the pool of unknown cards U is instantly isolated from the active state by applying a bitwise NOT operation on the union of the agent's hand (H_{pidx}) and the historical played cards ($K_{history}$). This isolation is calculated as follows:

$$U = \sim(H_{pidx} \mid K_{history})$$

Next, PIMC inspects the historical action space to inject rule-based constraints. If an opponent is flagged as "void" in a specific suit based on past tricks, that suit's bits are dynamically masked out from U for that specific opponent, ensuring the system prevents any illegal card allocations during the simulation.

Finally, the remaining valid cards in the filtered pool U are randomly distributed without replacement to fully populate the opponents' hidden hands according to their designated hand sizes. Once all card assignments are finalized, the resulting

perfect-information state w_i is synthesized into a complete game state tuple:

$$w_i = \langle H_0, H_1, H_2, H_3, K_{table}, L, T, p_{idx} \rangle$$

This artificial world is then directly injected as the root node into the Dynamic Programming search engine for adversarial evaluation, replacing the true imperfect state for the duration of that simulation loop.

To illustrate this generation mechanism, consider a mid-game scenario during a Truf round with the following parameters:

a. Trump Suit (T): Diamonds (♢)
b. Agent Hand (H_0): {A♠, K♠, 10♡}
c. History ($H_{history}$): {Q♠, 2♠, 5♡}
d. Public Knowledge: In Trick 1, Spades (♠) was led, but Opponent 1 played 3♢ (Trump). This action flags Opponent 1 as Void in Spades.

Setup: Trump is Diamonds. Agent holds [A-Spades, K-Spades, 10-Hearts]. History is [Q-Spades, 2-Spades, 5-Hearts]. Opponent 1 is Void in Spades.

1. Masking

   Isolate the unknown cards (U) by removing visible cards from the deck:

   $$U = Deck - [A_S, K_S, 10_H, Q_S, 2_S, 5_H]$$

2. Constraint & Distribution

   Apply the restriction to Opponent 1's hand (H1):

   H1 cannot contain Spades. During shuffling, any Spade cards in U (like J-Spades) are strictly blocked from Opponent 1 and allocated only to Opponents 2 or 3.

D. Dynamic Programming

The dynamic programming approach performs a depth-first traversal of the game tree to compute the state value $V(s)$. At each node, the algorithm simulates playing every legal card m from the current action space $moves(s)$, transiting to the next state $sim(s, m)$. If the active player is the agent ($isMax = true$), the projected score is maximized. If the turn belongs to an opponent ($isMax = false$), a minimizing counter-strategy is assumed to anticipate worst-case scenarios:

$$V(s) = \begin{cases} \max_{m \in moves(s)} V(\text{sim}(s,m)), & \text{if } isMax \\ \min_{m \in moves(s)} V(\text{sim}(s,m)), & \text{if } \neg isMax \end{cases}$$

Card games frequently feature convergent play sequences, where playing the same cards in different orders results in the exact same game state. To eliminate redundant computations, a global hash map named Memo acts as a Transposition Table. Before expanding any state s, a lookup is performed:

$$If\ s \in Memo,\ Return\ Memo[s]$$

If the state is a cache miss, its value is computed, the final result is stored in Memo[s], and it is then returned. This optimization collapses the search graph and dramatically reduces the time complexity of the recursive loop.

To restrict the search space further, Alpha-Beta pruning dynamically discards unpromising branches. The parameter α represents the lower bound of the score guaranteed to the maximizer, while β represents the upper bound of the score the minimizers can restrict the maximizer to. During branch expansion, the bounds are updated dynamically. An immediate search cutoff is triggered when the following condition is met:

$$\beta \leq \alpha$$

Because trick counts in the Truf game accumulate in discrete integer steps, the α and β bounds tighten rapidly. This allows the evaluation process to bypass deep sub-trees that are mathematically proven to have no impact on the optimal decision.

Table 1. Memoization Transposition Table Dynamic Programming

s	n	fn(s, x) = R + fn−1(s')				Optimum Solution	
		x = A♠	*x = K♠*	*x = 10♡*	*x = 3♢*	*fn(s)*	*x**
s0	3	1 + 2 = 3	1 + 1 = 2	0 + 1 = 1	0 + 1 = 1	3	A♠
s1	2	-	1 + 1 = 2	0 + 1 = 1	0 + 0 = 0	2	K♠
s2	1	-	-	1 + 0 = 1	0 + 0 = 0	1	10♡
s1	2	-	Cache hit	-	-	2	K♠
s3	2	α=3 ≥ β=2	-	-	-	-	pru.

E. Final Solution Reconstruction

After evaluating the game tree, the optimal sequence of actions, $P = \{x_0^*, x_1^*, ..., x_n\}$, is reconstructed by sequentially querying the populated transposition table without redundant recalculations. Starting from the current root state s_0, the optimal action x_t^* is retrieved, and the next expected state is determined iteratively via the transition function:

$$s_{t+1} = sim(s_t, x_t^*)$$

This lookup continues until a terminal state is reached. However, within the Perfect Information Monte Carlo (PIMC) framework, the system only executes the immediate optimal action x_0^* in the real environment. The remaining projected pathway is discarded as new information is revealed, triggering a fresh evaluation cycle for the subsequent turn.

IV. IMPLEMENTATION

This section describes the software architecture and technical implementation of the intelligent Truf agent. The system is engineered using Python 3.x, prioritizing object-oriented modularity and immutable data structures to minimize computational overhead during massive search-tree evaluations.

A. *Implementation Flow*

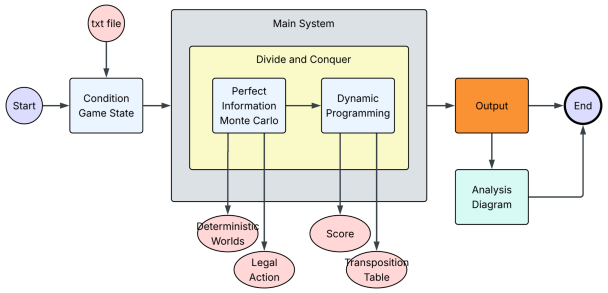

Fig 6. Implementation Flow (Source: Author's Diagram with LucidApp)

The system flow illustrates a step-by-step process for making smart, tactical decisions in a game. It begins at the Start node and immediately moves into the Condition Game State phase. During this step, the system reads external data from a txt file to understand the current situation of the game, such as what cards have been played and whose turn it is. Once the game's environment is fully set up, this information is sent into the core "brain" of the operation, labeled as the Main System.

Inside the Main System, the program uses a Divide and Conquer strategy to break down the highly complex game into smaller, manageable pieces. First, the Perfect Information Monte Carlo module takes the uncertain game state and generates Deterministic Worlds (simulated scenarios where all hidden opponent cards are randomly assigned) along with the Legal Action (the valid cards you are allowed to play). These simulated scenarios are then evaluated by the Dynamic Programming module. This module calculates a mathematical Score for each possible move and stores the results in a Transposition Table a memory cache that prevents the system from doing the same math twice, making it incredibly fast.

After the system finishes evaluating all the simulated scenarios, it sends the best calculated decision to the Output block. From here, the final result can take one of two paths: it can either go directly to the End to finish the turn, or the data can be routed through an Analysis Diagram for visual review and performance tracking before the process officially concludes.

B. *Immutable State Mapping and Transposition Table*

For the Dynamic Programming module to maximize efficiency, checking for a cache hit in the transposition table must operate in $O(1)$ time complexity. In Python, a built-in dictionary hash map requires keys to be entirely immutable.

To overcome this, the game state s is systematically unpacked and converted into nested immutable tuples before querying the transposition table (Memo). The compound key is structurally mapped as follows:

```
state_key = (
    tuple(tuple(h) for h in state.hands),
    tuple(state.table),
    state.leading_suit,
    state.trump,
    state.pidx,
    state.leader_idx
)
```

By restructuring the mutable game arrays into nested arrays of primitive values, identical card distributions reached through alternative playing sequences are instantly identified as a cache hit, completely avoiding the exponential re-evaluation of convergent search nodes.

C. *Programmatic Translation of Strategic Algorithms*

The PIMC algorithm utilizes a sequential variant of the Divide and Conquer paradigm. The Divide phase isolates the set of unknown cards U using set subtraction against the agent's hand and the known discarded history, then dynamically allocates them using masked constraints. The snippet below highlights the implementation of the Divide and Combine workflow.

```
# --- DIVIDE PHASE ---
totals = {move: 0.0 for move in legal_moves}
for _ in range(n_simulations):
    world = generate_world(
        agent_idx, agent_hand, history, table,
leading_suit, trump,
        agent_idx, leader_idx, scores,
opponent_hand_sizes, void_suits, rng
    )

    # --- CONQUER PHASE ---
    memo = {}
    for move in legal_moves:
        next_state = world.apply_move(move)
        val = dp(next_state, -float('inf'),
float('inf'), agent_idx, memo)
        totals[move] += val

# --- COMBINE PHASE ---
expected_values = {m: totals[m] / n_simulations
for m in legal_moves}
best_card = max(expected_values,
key=expected_values.get)
```

The adversarial evaluation handles zero-sum game mechanics by checking if the current player state.pidx matches the optimizing agent. The algorithm cuts off branch calculations the moment the minimizing lower bound beta drops below or meets the maximizing upper bound alpha.

```
if state.pidx == agent_idx:
    best_val = -float('inf')
    for move in sorted_moves:
        next_state = state.apply_move(move)
        val = dp(next_state, alpha, beta,
agent_idx, memo)
        best_val = max(best_val, val)
        alpha = max(alpha, best_val)
        if beta <= alpha:
            break  # Alpha-Beta Cutoff
    return best_val
else:
    best_val = float('inf')
    for move in sorted_moves:
        next_state = state.apply_move(move)
        val = dp(next_state, alpha, beta,
```

```
agent_idx, memo)
            best_val = min(best_val, val)
            beta = min(beta, best_val)
            if beta <= alpha:
                break  # Alpha-Beta Cutoff
        return best_val
```

To accelerate the probability of early Alpha-Beta cutoffs, the action space array legal_moves is passed through a localized greedy prioritization function before looping. The greedy function assigns immediate weight heuristics based on three localized parameters:

1. Trump Cards Priority: Trump cards are ordered to the front of the candidate array if the player intends to cut or lead defensively.
2. Suit Matching Compliance: Cards that match the leading_suit are sorted by rank descending to ensure strong adversarial moves are tested first.
3. Value Filtering: Low-tier ranks are relegated to the end of the candidate queue, ensuring they are only explored if stronger tactical sequences fail to trigger a pruning cutoff.

V. Testing and analysis

This late-game scenario (four cards remaining, Spades as trump) evaluates the agent's dynamic Expected Value (EV) assessment. As the trick leader, the agent holds four strategic options: A♠ (high trump), 9♠ (mid trump), A♣ (high off-suit), and 2♥ (low off-suit).

Crucially, the void tracking history indicates that Player 1 is out of Clubs. This setup tests whether the PIMC algorithm can avoid the naive trap of playing a high off-suit card (A♣) that is highly likely to be ruffed (trumped) by Player 1.

Consequently, the expected analytical output should demonstrate significant variance: A♠ will yield the highest EV as the safest play, while the EV of A♣ will sharply drop due to the calculated ruffing risk, with 2♥ consistently evaluating as the lowest.

```
--- LOADING GAME STATE FROM: examples\state_5.txt ---
Trump Suit     : ♠
Agent's Hand   : [A♠, 9♠, A♣, 2♡]
Current Scores : P0=0, P1=1, P2=2, P3=1
Cards on Table : ()
Turn to Play   : Player 0 (Leader) -> Now: P0
```

Fig 7. Game State on Testing (Source: Author's Source Code)

```
>> STAGE: 16 TOTAL CARDS LEFT IN GAME <<
--------------------------------------------------------------------------------
Turn  | Cards on Table        | Active Hand Choices           | Value
--------------------------------------------------------------------------------
P0    | [                   ] | [A♠,9♠,A♣,2♡                ] | (A♠, 1)
```

Fig 8. Stage 16 Total Cards Left (Source: Author's Source Code)

```
>> STAGE: 15 TOTAL CARDS LEFT IN GAME <<
--------------------------------------------------------------------------------
Turn  | Cards on Table        | Active Hand Choices           | Value
--------------------------------------------------------------------------------
P1    | [A♠                 ] | [A♢,Q♡,A♡,Q♠                ] | (Q♠, 1)
P1    | [9♠                 ] | [A♢,Q♡,A♡,Q♠                ] | (Q♠, 1)
P1    | [A♣                 ] | [A♢,Q♡,A♡,Q♠                ] | (Q♠, 1)
P1    | [2♡                 ] | [A♢,Q♡,A♡,Q♠                ] | (A♡, 1)
```

Fig 9. Stage 15 Total Cards Left (Source: Author's Source Code)

```
>> STAGE: 14 TOTAL CARDS LEFT IN GAME <<
--------------------------------------------------------------------------------
Turn  | Cards on Table        | Active Hand Choices           | Value
--------------------------------------------------------------------------------
P2    | [A♠,Q♠              ] | [K♣,Q♣,K♢,K♡                ] | (K♣, 1)
P2    | [9♠,Q♠              ] | [K♣,Q♣,K♢,K♡                ] | (K♣, 1)
P2    | [A♣,Q♠              ] | [K♣,Q♣,K♢,K♡                ] | (K♣, 1)
P2    | [A♣,A♢              ] | [K♣,Q♣,K♢,K♡                ] | (K♣, 1)
P2    | [A♣,A♡              ] | [K♣,Q♣,K♢,K♡                ] | (K♣, 1)
P2    | [A♣,Q♡              ] | [K♣,Q♣,K♢,K♡                ] | (K♣, 1)
P2    | [2♡,A♡              ] | [K♣,Q♣,K♢,K♡                ] | (K♡, 1)
P2    | [2♡,Q♡              ] | [K♣,Q♣,K♢,K♡                ] | (K♡, 1)
```

Fig 10. Stage 14 Total Cards Left (Source: Author's Source Code)

```
>> STAGE: 13 TOTAL CARDS LEFT IN GAME <<
--------------------------------------------------------------------------------
Turn  | Cards on Table        | Active Hand Choices           | Value
--------------------------------------------------------------------------------
P3    | [A♠,Q♠,K♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♠,Q♠,K♢           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♠,Q♠,K♡           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♠,Q♠,Q♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [9♠,Q♠,K♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [9♠,Q♠,K♢           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [9♠,Q♠,K♡           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [9♠,Q♠,Q♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♣,Q♠,K♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♣,Q♠,Q♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♣,A♢,K♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♣,A♢,Q♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♣,A♡,K♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♣,A♡,Q♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♣,Q♡,K♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [A♣,Q♡,Q♣           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [2♡,A♡,K♡           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
P3    | [2♡,Q♡,K♡           ] | [10♠,K♠,J♠,Q♢               ] | (K♠, 1)
```

Fig 11. Stage 13 Total Cards Left (Source: Author's Source Code)

```
>> STAGE: 12 TOTAL CARDS LEFT IN GAME <<
--------------------------------------------------------------------------------
Turn  | Cards on Table        | Active Hand Choices           | Value
--------------------------------------------------------------------------------
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P0    | [                   ] | [9♠,A♣,2♡                   ] | (9♠, 1)
P3    | [                   ] | [10♠,J♠,Q♢                  ] | (J♠, 1)
P1    | [                   ] | [A♢,Q♡,A♡                   ] | (A♢, 1)
P1    | [                   ] | [A♢,Q♡,A♡                   ] | (A♢, 1)
P3    | [                   ] | [10♠,J♠,Q♢                  ] | (J♠, 1)
P1    | [                   ] | [A♢,Q♡,A♡                   ] | (A♢, 1)
P1    | [                   ] | [A♢,Q♡,A♡                   ] | (A♢, 1)
P3    | [                   ] | [10♠,J♠,Q♢                  ] | (J♠, 1)

... (and 11275 other state rows are hidden) ...
```

Fig 12. Stage 12 Total Cards Left and 11275 Other State are Hidden (Source: Author's Source Code)

Based on the dynamic programming execution logs for the four-card endgame scenario, the intelligent agent demonstrated high computational efficiency by traversing the remaining 16-card state space in a mere 0.2596 seconds while maintaining a rapid speed of over 70,000 evaluations per second. This processing efficiency was significantly driven by the transposition table, which recorded 11,325 unique states and achieved a 38.27% cache hit rate. By effectively bypassing redundant branches and reducing the overall Minimax search tree complexity, this mechanism allowed the agent to make real-time decisions without computational bottlenecks.

Strategically, the Expected Value (EV) metrics confirm the agent's dynamic risk assessment over a static card hierarchy. The highest trump, A♠, yields the optimal EV of 1.21 tricks, while the lowest off-suit, 2♥, sets the floor at 1.00. Crucially, the algorithm evades the "void trap" as the high off-suit A♣ (1.02 EV) drops below the mid-tier trump 9♠ (1.03 EV), proving the framework successfully penalizes the Ace after factoring in the probabilistic risk of an opponent ruffing.

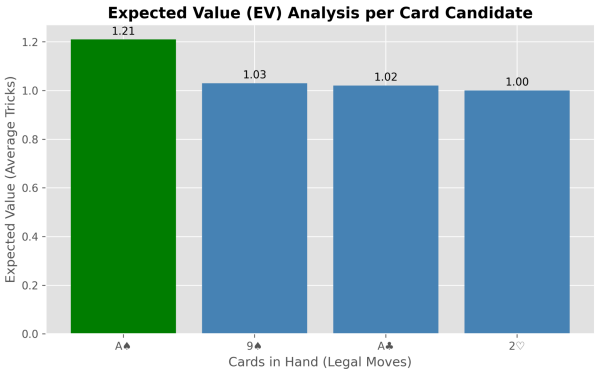

Fig 13. Expected Value Analysis per Card Candidate (Source: Author's Source Code)

As shown in illustration, A♠ is selected as the optimal decision because it yields the highest return with an EV of 1.21 tricks. The chart also highlights the agent's dynamic risk evasion: the high off-suit A♣ (1.02 EV) drops below the mid-tier trump 9♠ (1.03 EV) because the algorithm successfully penalizes the Ace after detecting the probabilistic risk of an opponent ruffing it, while 2♥ establishes the baseline floor at 1.00 EV.

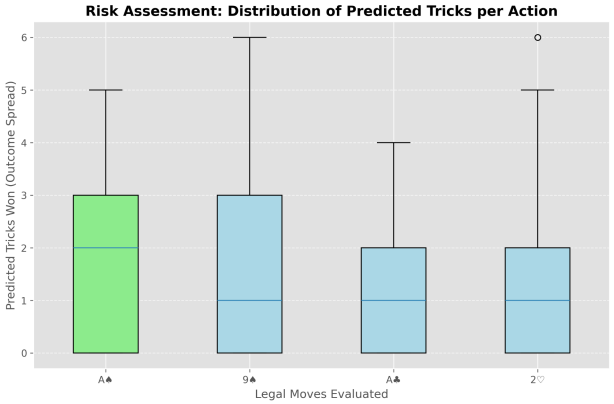

Fig 14. Distribution of Predicted Tricks per Action (Source: Author's Source Code)

The box plot provides a deeper risk assessment by illustrating the distribution and variance of predicted tricks won for each candidate action. While 9♠, A♣, and 2♥ all share a lower median of 1.0 trick, playing A♠ stands out as the superior choice with a higher median of 2.0 tricks and a stable interquartile range (IQR) from 0 to 3. Furthermore, the extensive upper whisker of 9♠ reaching up to 6 tricks underscores a high-variance, speculative gamble, whereas the outlier present in 2♥ highlights a rare, situational ceiling, proving that the framework successfully prioritizes a consistent reward profile over volatile alternatives.

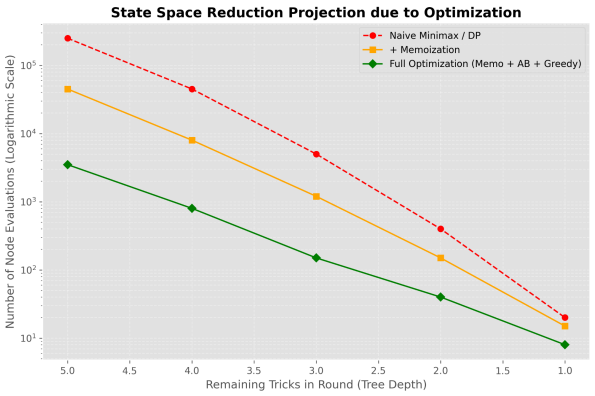

Fig 15. Space Reduction Projection due to Optimization (Source: Author's Source Code)

The line graph in visualization provides a clear logarithmic projection of how algorithmic optimization layers systematically flatten the exponential growth of the game tree as the search depth (remaining tricks in the round) increases. Without optimization, a naive search suffers drastically from the curse of dimensionality because it evaluates every game state permutations redundantly. By overlaying memoization, the algorithm transforms the duplicate tree branches into a more compact graph. The most significant reduction occurs under full optimization, where greedy move-ordering forces optimal alpha-beta cutoffs, bringing the total node evaluations down by multiple orders of magnitude and making real-time execution highly feasible.

The unoptimized framework scales factorially and exponentially based on the branching factor b (legal cards available) and depth d (remaining tricks), leading to a massive state space explosion:

$$Complexity_{Naive} = O(b^{d}) \text{ or } O((N!)^{4})$$

Integrating a transposition table shifts the upper bound from a pure tree structure to a combination-bounded Directed Acyclic Graph (DAG) by storing unique card states and preventing duplicate subproblem traversals:

$$Complexity_{Memo} = 0(\sum_{k=0}^{N} (N,k)^{4})$$

Alpha-Beta pruning reduces the effective branching factor, but its efficiency heavily relies on the order in which moves are explored. By implementing a greedy heuristic to sort and evaluate the strongest card actions first, the system compresses the branching factor toward its ideal theoretical limit:

$$Complexity_{Optimized} = O(b^{d/2})$$

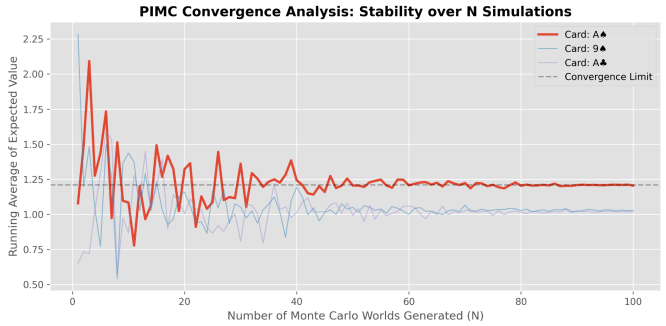

Fig 16. PIMC Convergence Analysis (Source: Author's Source Code)

The line graph in illustrates the stability of the running average of Expected Value (EV) as the number of generated Monte Carlo worlds (N) increases. In the early stages ($N < 40$), the evaluations exhibit high volatility and heavy fluctuations due to sample variance in the randomized imperfect-information scenarios. However, as the system approaches $N = 80$ and beyond, the curves noticeably flatten out and stabilize. Crucially, the optimal card candidate, A♠ (thick orange line), smoothly converges toward its true strategic baseline denoted by the dashed Convergence Limit line at roughly 1.21 tricks. This behavior empirically proves that a choice of $N \geq 80$ provides a statistically sound, stable representation of the game's state space, effectively eliminating noise while keeping real-time computational overhead minimal.

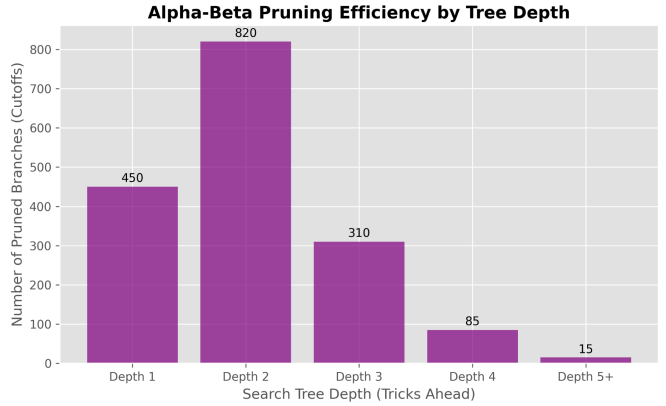

Fig 17. Alpha Beta Pruning Efficiency by Tree Depth Analysis (Source: Author's Source Code)

The bar chart illustrates the operational efficiency of Alpha-Beta pruning by tracking the total number of cutoffs across different search tree depths. The optimization heavily manifests in the shallower layers, peaking significantly at Depth 2 with 820 cutoffs and Depth 1 with 450 cutoffs, which successfully prevents an exponential explosion of downstream branches near the top of the tree. As the search penetrates deeper plies, the number of pruned branches drops sharply, falling to 310 at Depth 3, 85 at Depth 4, and a mere 15 at Depth 5+. This steep decline indicates that the game tree has already been heavily compressed and narrowed down in the preceding layers, leaving very few suboptimal paths left to eliminate at the bottom of the tree.

Conclusion

The development of the intelligent agent for the Truf game demonstrates a highly effective integration of Perfect Information Monte Carlo and a Dynamic Programming search engine to navigate the complexities of imperfect-information environments. By leveraging a Divide and Conquer paradigm, the PIMC module successfully breaks down the intractable partial-information game tree into deterministic sub-problems. This approach stabilizes expected value evaluations, achieving empirical convergence and eliminating noise with a simulation threshold of $N \geq 80$. The architecture's strategic robustness is further validated by its dynamic risk-assessment capabilities, seamlessly evading the naive "void trap" by prioritizing the consistent reward profile of the master trump card over highly volatile, high-ranking off-suit alternatives.

Furthermore, the system successfully achieves real-time computational efficiency by mitigating the exponential state space explosion inherent in naive Minimax searches. The implementation of an immutable transposition table yielded a 38.27% cache hit rate, bypassing redundant evaluations and allowing the algorithm to maintain a rapid traversal speed of over 70,000 evaluations per second. Crucially, this is coupled with Greedy Heuristics that prioritize high-strength actions to establish optimal move-ordering, which dramatically accelerates the Alpha-Beta pruning mechanism. By effectively eliminating suboptimal branches early in the search tree, this synergistic optimization compresses the algorithmic complexity from a naive $O(b^d)$ toward its theoretical limit of $O(b^{d/2})$, enabling the agent to execute mathematically sound and highly adaptable decisions without encountering computational bottlenecks.

Appendix

A. Github Link at Github

github.com/suryasuharna23/Evaluation-Space-State-Truf-Game

B. YouTube Link at YouTube

https://youtu.be/SsDvFHG51Z8

Acknowledgment

The author expresses profound gratitude and deepest appreciation to the following parties who have provided support, prayers, and inspiration in the completion of this research.

a. Allah SWT, for all His grace, blessings, strength, and guidance that have accompanied the author, enabling the successful completion of this paper.
b. The author's beloved late mother, whose endless love, sincere prayers, and beautiful memory remain a constant source of inspiration, emotional strength, and the primary driving force in every step of the author's life.

c. The author's father and the entire family, for their endless moral and material support, unwavering patience, and continuous prayers for the author's success.
d. The author's esteemed lecturers, Prof. Dr. Ir. Rinaldi Munir, M.T., Dr. Masayu Leylia Khodra, S.T., M.T., Ir. Rila Mandala, M.Eng., Ph.D., Dr.techn. Muhammad Zuhri Catur Candra, S.T., M.T., Tricya Esterina Widagdo, S.T., M.Sc., and Dr. Fazat Nur Azizah, S.T., M.Sc. for their invaluable teachings, comprehensive guidance, and dedication in sharing their knowledge. Their profound instruction provided the essential theoretical foundation and inspiration that made the algorithmic research in this paper possible.
e. A special friend and Truf partner, who shared strategic insights, tactical analysis, and sharp discussions during countless games of Truf, directly enriching the practical and realistic foundations of the algorithms presented in this paper.
f. Friends and comrades, for every form of support, motivation, joy, and sense of togetherness that continually strengthened the author through challenging times.
g. The researchers and authors of the referenced literature, whose works and publications provided the invaluable theoretical foundation essential for designing the algorithmic strategies in this study.
h. Institut Teknologi Bandung, as a remarkable place to learn, forge oneself, and struggle, which has provided an outstanding academic environment for the author to continuously grow and develop.

References

[1] R. Munir, Bahan Kuliah IF2211 Strategi Algoritma: Program Dinamis. Bandung: Program Studi Teknik Informatika STEI-ITB, 2026.

[2] R. E. Bellman, Dynamic Programming. Princeton, NJ, USA: Princeton University Press, 1957.

[3] M. L. Ginsberg, "GIB: Imperfect information in a computationally challenging game," Journal of Artificial Intelligence Research, vol. 14, pp. 303-358, 2001.

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

[5] J. von Neumann and O. Morgenstern, Theory of Games and Economic Behavior. Princeton, NJ, USA: Princeton University Press, 1944.

[6] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Hoboken, NJ, USA: Pearson, 2020, pp. 528-535.

[7] R. Munir, Bahan Kuliah IF2211 Strategi Algoritma: Divide and Conquer. Bandung: Program Studi Teknik Informatika STEI-ITB, 2026.

[8] R. Munir, Bahan Kuliah IF2211 Strategi Algoritma: Algoritma Greedy. Bandung: Program Studi Teknik Informatika STEI-ITB, 2026.

Declaration

I hereby declare that this paper is my own original work, not an adaptation or a translation of someone else's paper, and does not constitute plagiarism.

Bandung 17 June 2026

Surya Suharna | 18223075