Deterministic Isomorphic Validation of Composite Card Structures in Non-Monotonic Rank Hierarchies

Muhammad Rengga Putra Kuncoro, and Rully Soelaiman, *Member, IAENG*

***Abstract*—Structural verification in multi-deck sequential games is a fundamental combinatorial problem where the objective is to validate player adherence to complex lead templates. Typical approaches to this problem require exploring vast search spaces, which creates a bottleneck when real-time arbitration is a crucial factor. In this paper, we propose a deterministic solution using a piecewise mapping transformation, Φ, and a greedy decomposition strategy. Our method projects discontinuous rank hierarchies into a normalized integer domain for constant-time adjacency checks and exploits physical domain constraints to minimize the cardinality of the requirement blocks. By simplifying the multiset matching problem into smaller sub-problems, the engine navigates the symmetric group S_n within strictly bounded limits. Empirical results show that this framework achieves deterministic rule enforcement within the sub-millisecond latency thresholds required for interactive gameplay. This solution ensures high efficiency in resource usage, providing a rigorous foundation for rule enforcement in complex combinatorial environments such as the game of Tractor.**

***Index Terms*—combinatorial optimization, structural isomorphism, non-monotonic hierarchies, greedy decomposition.**

I. Introduction

THE computational modeling of multi-deck trick-taking systems represents a significant departure from standard card game logic. While traditional games rely on a static, linear ordinality, complex combinatorial environments necessitate a transition from simple scalar comparisons to a multidimensional analysis of play structure. The primary difficulty lies in the fact that the game state is not merely a collection of values but a structural topology that must be preserved across player turns. This research addresses the fundamental hurdles of this domain: the resolution of rank discontinuities and the verification of structural isomorphism in high-entropy environments.

The first challenge involves the fluid nature of power hierarchies. In these environments, the introduction of dynamic variables such as a floating Current Rank and multi-tiered trump partitions induces a structural discontinuity in the standard sequence. This creates a broken hierarchy where the traditional adjacency of cards is severed. Without a robust method to normalize this irregular rank-space, a system cannot reliably verify sequential patterns because the physical distance between cards in a deck no longer correlates to their mathematical distance in the game logic.

Beyond the ranking of individual cards, a second challenge emerges during the execution of a Throw, which will be discussed in Chapter 2. When a player leads multiple structures simultaneously, such as nested pairs or complex sequences, the problem evolves into a search for structural equivalence. The arbiter must determine if a follower's hand can be partitioned into a subset that satisfies the lead's template, regardless of the specific cards used. This process requires navigating the significant overhead inherent in generating all permutations of a player's hand to find a valid response, a task that approaches the complexity of a graph isomorphism check [3] [4]. Ensuring that the internal relationships between cards in a play are preserved under a mapping that reflects the current game state is essential for maintaining the logical integrity of the engine.

Inspired by the `TRACTOR` challenge on the Sphere Online Judge (SPOJ) platform [1], this paper defines a formal method for verifying these plays. We propose an integrated framework that first flattens the rank hierarchy and then decomposes complex plays into their constituent primitives. This allows for rigorous rule enforcement and dominance evaluation in multi-deck environments with bounded multiplicity. The proposed system achieves this through a three-stage process:

i. Transforming the irregular, discontinuous rank-space into a normalized, continuous integer domain (Φ) for efficient adjacency verification.
ii. Implementing a greedy structural deconstruction engine that decomposes complex plays into their constituent primitives.
iii. Applying a matching algorithm to verify structural isomorphism between the lead's requirements and the follower's hand.

Manuscript received January 06, 2026; This work was supported in part by Sepuluh Nopember Institute of Technology, Surabaya, Indonesia.

Muhammad Rengga Putra Kuncoro is an undergraduate student at Sepuluh Nopember Institute of Technology, Department of Informatics, Surabaya, Indonesia (e-mail: muhammadrenggaaputrakuncoro@gmail.com)

Rully Soelaiman is an associate professor at Sepuluh Nopember Institute of Technology, Department of Informatics, Surabaya, Indonesia (e-mail: rully130270@gmail.com)

This integrated framework[1] ensures that even the most complex strategic maneuvers are validated against a sound mathematical foundation.

II. Problem Statement

In this section, we formalize the structural constraints of the environment to establish a rigorous basis for algorithmic validation. We define the game state by decomposing the deck composition, establishing the rank hierarchy through mapping functions, and identifying the specific requirements for isomorphic adherence between leading and following plays.

To bridge the gap between abstract game rules and computational logic, we address the non-monotonic nature of the card hierarchy, where the relative value of a card is not fixed but fluctuates based on the trump selection. By treating a hand not as a simple list of values but as a collection of potential topological structures, we can model the arbitration process as a search for structural equivalence. This section details the mathematical transformation of the 108-card double deck into a normalized priority space, the classification of card multisets into discrete structural primitives, and the permutative conditions that determine the legality of a play within the S_n search space.

A. Deck Composition and Bounded Multiplicity

The system operates on a multi-deck environment where each card c is defined by the tuple (r, s), representing its rank and suit respectively. In a two-deck configuration, the multiplicity of any unique card is exactly two. This bounded multiplicity is a critical constraint, as it facilitates the formation of identical pairs which serve as the fundamental building blocks for higher-order structures.

B. The Active Hierarchy and Trump System

Unlike standard trick-taking games, the rank hierarchy is dynamic and non-linear. A Current Rank $CR \in \{2, 3, \ldots, 14\}$ and a Main Suit S_m are designated at the start of each round. For the purpose of mathematical modeling, face cards are mapped to integer values as follows:

TABLE I
Face Card Rank Mapping

Rank Symbol	J	Q	K	A
Integer Value	11	12	13	14

The designation of CR and S_m partitions the deck into two primary sets:

- **Trump Set ($\mathcal{T}$):** Includes all cards of suit S_m, all cards of rank CR regardless of suit, and the supreme trumps (Jokers).
- **Natural Set ($\mathcal{N}$):** Includes all remaining cards, organized by their respective suits.

A critical aspect of this system is *Rank Displacement*. The CR is extracted from its natural sequence and elevated to a priority position within $\mathcal{T}$ [1]. This elevation creates a structural discontinuity in the remaining ranks. For instance, if $CR = 11$ (Jack), then the ranks 10 and 12 become logically adjacent.

C. Structural Primitives: Pairs and Tractors

The game recognizes three levels of structural complexity, each defined by its multiplicity and adjacency requirements.

1) **Singles ($k = 1$):** Any individual card $c \in \mathcal{C}$.
2) **Pairs ($k = 2$):** A set of two identical cards $P = \{c_1, c_2\}$ such that $rank(c_1) = rank(c_2)$ and $suit(c_1) = suit(c_2)$.
3) **Tractors ($L \geq 2, k = 2$):** A sequence of L pairs.

Formally, we define a Tractor T as a collection of cards that must satisfy a strict homogeneity constraint [1]. Let $S(c)$ be a function returning the suit of card c, and let $\mathcal{T}$ be the Trump set. A Tractor T of length L is valid if and only if:

$$\forall c_i, c_j \in T, \quad (c_i, c_j \in \mathcal{T}) \vee (S(c_i) = S(c_j) = \mathcal{S}_n)$$

where $\mathcal{S}_n$ is a specific natural suit. Furthermore, the set of unique transformed ranks $\{\Phi(c) \mid c \in T\}$ must form a contiguous integer interval:

$$\{\Phi(c) \mid c \in T\} = \{x, x+1, \ldots, x+L-1\}$$

D. Composite Structural Sets (The Throw)

In a game of Tractor (Sheng Ji), a "Throw" is defined as an *atomic composite move* comprising a multi-set of cards C [1]. Unlike standard moves, a Throw is a batch operation where the entire set must be confined to a single logical partition of the deck to maintain suit-integrity constraints. Formally, C is a valid composite structure if and only if:

$$C \subseteq \mathcal{T} \quad \vee \quad \exists \mathcal{S}_n \in \{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\} : C \subseteq \mathcal{S}_n$$

where $\mathcal{T}$ is the Trump set and $\mathcal{S}_n$ is a natural suit partition.

From a validation perspective, a Throw represents a *compound requirement template*. While it is played as a single unit, the underlying verification engine must treat it as a collection of independent structural blocks $\mathcal{B}_L$ that the follower must match isomorphically. This introduces a complexity spike, as the engine must verify not just the existence of the cards, but the adherence to the internal structural density (pairs and sequences) within the batch.

[1]The complete C++ implementation, Python data analysis, and documented LaTeX source are available at: https://github.com/tendonintendo/TRACTOR

E. Isomorphic Adherence

The primary challenge in Tractor arbitration is the requirement of isomorphic adherence. When a lead play P_L is executed, it establishes a template of structural blocks. A following play P_F must match this template if the player's hand contains the requisite structures. The complexity arises from the fact that a single hand can be partitioned into multiple valid multisets of blocks; the algorithm must determine if any such partition exists that satisfies the lead's template.

In this section, we visualize the decomposition path where a follower's superior structure is sliced to meet a lower order requirement. For instance, if P_L requires two discrete pairs, a follower holding a tractor of length $L = 2$ can successfully satisfy the lead through structural degradation. This formalization allows us to treat the validation of a play as a search problem within the symmetric group S_n, where the goal is to find a mapping that satisfies the structural requirements of the lead while respecting the resource constraints of the follower's hand.

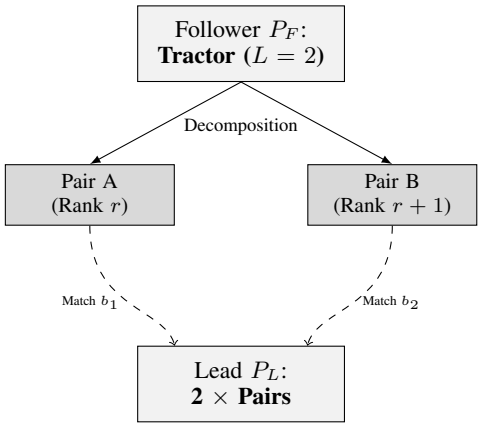

Fig. 1. Structural decomposition of a Tractor into two Pairs

By framing the hand validation through this decomposition lens, the algorithm ensures that players are held to their highest structural obligations. If the condition fails for all permutations $\sigma \in S_n$, it indicates that no valid interpretation of the hand can satisfy the lead template, resulting in a non-conforming play.

III. Methodology

Our methodology addresses the complexity of multi-deck arbitration by implementing a sequential pipeline, as illustrated in Fig. 2, that first linearizes the state space and then executes a structural search. This approach is necessitated by the fact that standard integer comparisons are no longer valid in games with dynamic hierarchies. By re-indexing the deck through a Rank Mapping Transformation, the system effectively neutralizes the structural discontinuities caused by the Current Rank and Trump partitions. As noted by Buro and Long, this mapping is the foundation for the "efficient play and analysis of trick-taking games," as it allows the engine to treat a fragmented hierarchy as a synthetic monotonic sequence [6].

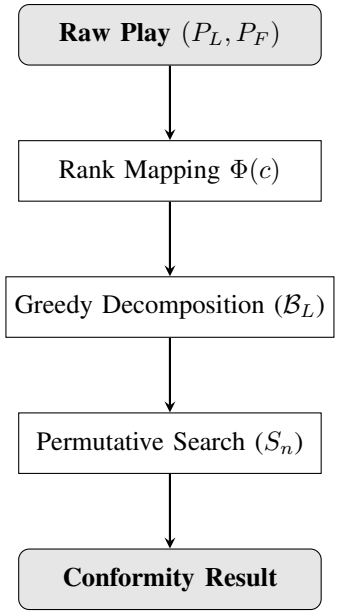

Fig. 2. The verification pipeline from input to trick arbitration

The second stage of our methodology moves from individual card values to the decomposition of the lead's template. When evaluating a Throw, the system treats the lead play as a structural requirement that the follower must satisfy. This necessitates a transition into a combinatorial search space where the arbiter must identify a valid partition of the follower's hand. As discussed in *Computers and Intractability: A Guide to the Theory of NP-Completeness* by Garey and Johnson, identifying disjoint subsets within a larger set mirrors the "Exact Cover" problem [2]. To navigate the overhead inherent in "generating all permutations" of card combinations [3], the engine utilizes the normalized rank domain to identify potential primitives—singles, pairs, and tractors—that can be matched against the lead's requirements.

To manage the exponential search space of a multi-deck hand, the engine employs a Greedy Partitioning Algorithm by prioritizing the largest and most complex structures first, such as long sequential tractors, before filling remaining requirements with simpler primitives. This ensures that the follower's response is architecturally identical to the lead template, effectively solving the structural equivalence problem without the computational cost of a full "graph isomorphism" check [4].

By interleaving these two processes, the system maintains logical integrity despite the high entropy of the multi-deck environment. The arbitration engine first flattens the rank-space to enable adjacency checks and then applies the greedy deconstruction to verify structural isomorphism. This dual-layered approach

provides a rigorous mathematical basis for validating legal plays and determining trick dominance in real time.

A. The Rank-Mapping Transformation $\Phi(c)$

To facilitate constant-time adjacency verification in a discontinuous rank-space, we define a transformation function $\Phi : C \to \mathbb{Z}^+$. This mapping projects each card c into a normalized integer domain where sequential structural primitives (Tractors) can be identified via simple arithmetic.

1) Mathematical Formalization: Let $p(c)$ be the raw precedence of a card rank based on the set $\{2, 3, \ldots, A\}$, mapped to $\{1, \ldots, 13\}$. The transformation is defined piecewise based on the card's membership in the Trump ($\mathcal{T}$) or Natural ($\mathcal{N}$) partitions:

1) **Trump Set $\mathcal{T}$:**
   - Supreme Trumps: $\Phi(c) \in \{27, 28\}$.
   - CR Parity: $\Phi(c) \in \{25, 26\}$ based on $suit(c) = S_m$.
   - Main Suit Residuals: $\Phi(c) = p(c) + 12 - \mathbb{1}(p(c) > p_{CR})$.
2) **Natural Set $\mathcal{N}$:**
   - Natural Adjacency: $\Phi(c) = p(c) - \mathbb{1}(p(c) > p_{CR})$.

2) Algorithmic Implementation: The following algorithm illustrates the normalization of card values, ensuring $O(1)$ adjacency verification across suit boundaries and rank displacements.

Algorithm 1: Rank Normalization $\Phi(c)$

Input: Card c, Current Rank CR, Main Suit S_m
Output: Normalized Rank Value Φ

```
p ← precedence(c.rank)
p_CR ← precedence(CR)
if S_m is active then
    if c.suit == 'R' ∨ c.suit == 'B' then
        return c.suit == 'R' ? 28 : 27
    else if c.rank == CR then
        return c.suit == S_m ? 26 : 25
    else if c.suit == S_m then
        return p + 12 − (p > p_CR ? 1 : 0)
// Natural suit or No-Trump logic
if p > p_CR then
    return p − 1
return p
```

B. Maximal Structural Decomposition

The validation of a composite play, often referred to as a "Throw," requires the reduction of a multiset of cards C into its most significant structural components. This is achieved through a Greedy Descending Decomposition algorithm, a method predicated on the observation that higher-order primitives represent the most constrained elements of the play. As noted in *Introduction to Algorithms* by Cormen et al., a greedy approach is viable when the problem exhibits the "greedy-choice property," where a locally optimal choice, in this case selecting the largest possible structure, contributes to a globally consistent partition [5].

This methodology is necessitated by the unidirectional recursive nature of the structural matching problem. In multi-deck environments, the hierarchy of primitives is strictly non-invertible. While a higher-order primitive can be logically decomposed to satisfy lower-order requirements during the validation phase, such as a tractor being treated as individual pairs or singles, the inverse is logically impossible. A collection of disparate singles cannot be synthesized to fulfill a sequence requirement. Therefore, the lead template must be defined by its maximal possible primitives to establish the absolute "upper bound" of structural strictness for the trick.

By prioritizing the extraction of tractors, then pairs, and finally singles, the system effectively manages the overhead inherent in "generating all permutations" of a player's hand [3]. This descending approach ensures that the most complex structural dependencies are resolved first, which is critical when the underlying game state approaches the complexity of a graph isomorphism check [4]. By isolating the maximal structures, the engine can verify that the follower's response is not only of sufficient power but is architecturally identical to the lead template. This maintains the logical integrity of the arbitration process even as the rank-space fluctuates.

Formally, the decomposition partitions C into three disjoint sets:

$$C = \mathcal{P}_T \cup \mathcal{P}_P \cup \mathcal{P}_S$$

The algorithm executes the following procedural steps:

1. **Tractor Extraction ($\mathcal{P}_T$):** The system performs a greedy search for the longest possible contiguous sequence of pairs. Let $V = \{\Phi(c) \mid c \in C, \text{count}(c) = 2\}$. The algorithm seeks the largest L such that a sequence $\{v, v+1, \ldots, v+L-1\} \subseteq V$ exists. Once a Tractor is identified, those cards are removed from the pool before searching for smaller L.
2. **Pair Identification ($\mathcal{P}_P$):** Any remaining cards with multiplicity $k = 2$ that were not part of a Tractor are categorized as independent Pairs.
3. **Singular Residuals ($\mathcal{P}_S$):** All remaining cards are categorized as Singles ($k = 1$).

Decomposition as Search Space Reduction: This maximal extraction serves as a form of Lexicographical Pre-processing. By identifying the largest possible blocks first, the system minimizes the cardinality n of the requirement multiset $\mathcal{B}_L$. This is a critical optimization for the subsequent permutative matching phase; since the follower's search space grows at $O(n!)$, reducing n through greedy decomposition ensures that even complex 25-card throws are reduced to a small number of high-order blocks. This keeps the effective search space computationally trivial while

maintaining the recursive flexibility to "degrade" these large blocks if the follower's resources are fragmented.

1) Algorithmic Implementation: The extraction process uses a state-tracking array to ensure no card is assigned to multiple blocks. The engine iterates through potential sequence lengths L in descending order, ensuring that Tractors are prioritized over independent Pairs and Singles.

Algorithm 2: Greedy Block Extraction (`getStructure`)

Input: Sorted card array C, Total cards n
Output: Structural Template $\mathcal{B}_L$

$used[1 \dots n] \leftarrow false$
`// Phase 1: Tractor Extraction (Length` $L \geq 4$`)`
for $len \leftarrow n$ ***down to*** 4 ***step*** -2 **do**
  for $i \leftarrow 0$ ***to*** $n - len$ **do**
    if $used[i \dots i + len - 1]$ *contains* $true$ **then**
      continue
    end
    if *isSequenceOfPairs(*$C[i \dots i + len - 1]$*)* **then**
      Add Tractor of length len to $\mathcal{B}_L$
      Mark $used[i \dots i + len - 1] \leftarrow true$
    end
  end
end
`// Phase 2: Pair Identification`
for $i \leftarrow 0$ ***to*** $n - 2$ **do**
  if $\neg used[i] \wedge \neg used[i+1] \wedge$ *isPair(*$C[i], C[i+1]$*)* **then**
    Inc. Pair count in $\mathcal{B}_L$
    $used[i], used[i+1] \leftarrow true$
  end
end
`// Phase 3: Singular Residuals`
Count all $used[j] == false$ as Singles in $\mathcal{B}_L$
return $\mathcal{B}_L$

C. Systemic Isomorphic Validation

Following the lead's play P_L, the system generates a multiset of required structural blocks $\mathcal{B}_L = \{b_1, b_2, \dots, b_n\}$ via maximal decomposition. The validation of the follower's play P_F is modeled as a search for a valid structural mapping. This problem is closely related to Permutation Pattern Matching (PPM), where the objective is to determine if a pattern permutation exists within a larger text permutation. As noted in the paper *Hardness of Permutation Pattern Matching* [7], the general case of pattern matching is NP-hard, yet certain classes, specifically those avoiding long grid like structures, can be resolved more efficiently. In the context of Tractor, P_F represents the multiset of cards which the system must attempt to partition to satisfy the pattern $\mathcal{B}_L$.

The complexity of this matching task is non-uniform and highly dependent on the density of the hand. While practical graph isomorphism testers perform efficiently on average cases, the paper *Constructing Hard Examples for Graph Isomorphism* demonstrates that it is possible to generate specific instances that are "difficult or even infeasible" for standard solvers. These hard examples are characterized by high Weisfeiler-Leman dimension and a lack of non-trivial automorphisms [9]. In our simulation, hands where $n \geq 16$ approximate these high complexity states, as the saturation of the rank hierarchy creates an abundance of overlapping structural possibilities that mirror the hard benchmarks described by Dawar and Khan [9].

Formal Existence Criterion: A play P_F is isomorphically adherent to P_L if and only if there exists at least one permutation of the required blocks $\sigma(\mathcal{B}_L)$ that can be successfully mapped to the cards in P_F. This is treated as a specialized case of the Graph Isomorphism (GI) problem, utilizing a piece patching approach where sub structures are matched to a target hand. The paper *Graph Isomorphism Algorithm using Pieces Patching Puzzle Technique (PPP-Technique)* [8] posits that for many complex graphs, "the algorithm could find the isomorphism of most large random graphs" by leveraging greedy sub graph matching. Formally, P_F is valid if:

$$\exists \sigma \in S_n : \forall j \in \{1, \dots, n\}, \quad \text{Match}(b_{\sigma(j)}, R_j) \neq \emptyset$$

where:

- S_n is the **Symmetric Group** of degree n, representing the set of all $n!$ possible permutations of the lead's structural blocks.
- $b_{\sigma(j)}$ is the j-th structural block (e.g., a Tractor of length 3) under the specific permutation σ.
- $R_j = P_F \setminus \bigcup_{k=1}^{j-1} m_k$ is the residual pool of cards remaining in the follower's hand after the first $j-1$ blocks have been satisfied.
- m_k is the specific subset of cards consumed from P_F to satisfy block $b_{\sigma(k)}$.

By iterating through S_n, the algorithm ensures that if there is any possible way to "slice" the follower's hand to match the lead's template, it will be found. This methodology mirrors the patching puzzle technique [8] by iteratively satisfying localized structural constraints to resolve the global isomorphic state. To mitigate the exponential overhead associated with hard instances [9], the engine employs greedy decomposition to prioritize the largest structural blocks, effectively pruning the search tree before the $n!$ combinatorial explosion occurs. If the condition fails for all $\sigma \in S_n$, the play is non-conforming, as no structural interpretation of the follower's hand satisfies the lead.

The Permutative Search Algorithm: The `sameStructure` engine serves as a non-deterministic arbitration layer designed to resolve structural ambiguity through a Recursive Backtracking Search over the Symmetric Group S_n. Unlike traditional greedy matchers that may fail due to premature resource commitment, where an early allocation of a pair might inadvertently break a potential longer tractor, this algorithm treats the follower's hand as a fluid resource pool. As illustrated in the permutative existence check search tree (Fig. 3), the system explores various orderings σ of the lead requirements $\{b_1, \dots, b_n\}$. By exploring

this permutation space, the engine ensures that every possible interpretation of the follower's cards is evaluated. This approach is fundamentally defined by *Structural Degradation* (Fig. 4), a process where complex, higher-order primitives like tractors are dynamically decomposed to satisfy simpler requirements like pairs or singles only when an exact match is unavailable. This methodology ensures the engine remains robust against the hard examples of graph isomorphism described by Dawar and Khan [9], where dense symmetries often defeat simpler heuristic solvers.

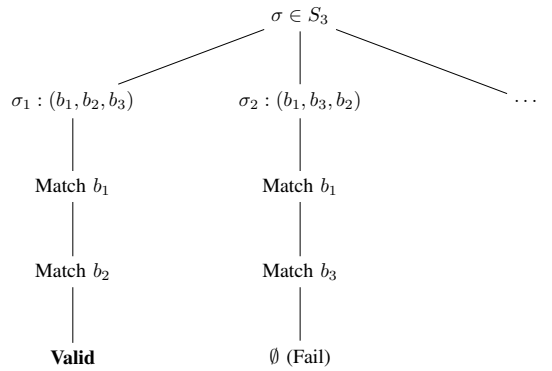

Fig. 3. Visualization of the permutative existence check search tree

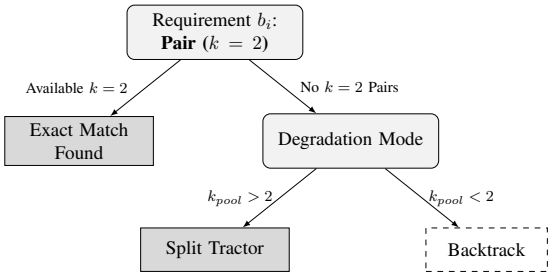

Fig. 4. Flow of the systemic isomorphic validation engine

1. **Search Space and Permutation Reset:** The algorithm iterates through permutations of the lead's template $\mathcal{B}_L$. For each new permutation $\sigma \in S_n$, the follower's card pool is fully restored to its original state. This ensures that every ordering of requirements, ranging from most specific structures like Tractors to most general ones like Singles, is tested against a complete resource set.
2. **Recursive Structural Degradation:** The core matching logic is inherently recursive. To satisfy a block b_i, the system executes a prioritized search:
   - **Exact Match:** If the current pool P_F contains a primitive identical to b_i in both count and rank continuity, it is consumed immediately to minimize search depth.
   - **Degradation (Recursive Case):** If an exact match is missing, the system attempts to break down a higher order primitive. For example, a Pair requirement may be satisfied by extracting two cards from a Tractor or a Triple. The consumed cards are removed from the active pool and the remaining fragments are returned as lower order residuals.
   - **Base Case (Singularization):** Any requirement can ultimately be satisfied by the base case of $k = 1$, provided the follower has sufficient card volume within the suit partition.
3. **Failure and Global Backtracking:** If a specific block b_i cannot be satisfied under the current permutation σ, the algorithm terminates that branch. It resets the follower's hand to its full initial state and proceeds to the next permutation $\sigma + 1$ in the search space. This mimics the piece patching methodology [8] by re-evaluating the global configuration when a local match fails.
4. **Existence Check:** If any branch of the recursion successfully satisfies all n requirements, the existence criterion is met. At this point the play is marked as Conforming and the search terminates immediately.

Conformity Failure: If the algorithm iterates through every $\sigma \in S_n$ and fails to find a single valid mapping, the play is deemed Non-Conforming. This result implies that no structural interpretation of P_F can satisfy the lead's template. In this state, the follower is disqualified from winning the trick, regardless of the rank values (Φ) of the cards played. This approach ensures that the "shape" of the play is treated as a hard logical constraint before any honor-based resolution occurs.

The function `sameStructure` iterates through these levels to confirm that P_F is a valid isomorphic representation of P_L. If the system determines that a follower's hand could have satisfied a higher-priority structural match but failed to do so, the play is flagged as non-conforming, and the player is excluded from the dominance resolution for that trick.

1) Algorithmic Implementation: The following procedure represents the core logic of the `sameStructure` engine. It utilizes a canonical sort of the lead blocks followed by a permutative expansion to ensure every possible structural interpretation is evaluated.

Algorithm 3: Permutative Matching Engine (`sameStructure`)

Input: Lead Template $\mathcal{B}_L$, Follower Template $\mathcal{B}_F$
Output: Boolean (Structural Conformance)

$S \leftarrow$ Linearize($\mathcal{B}_L$) `// Combines Tractors, Pairs, and Singles`
Sort S in descending order to establish a canonical starting permutation
repeat
  $(T_{pool}, P_{pool}, S_{pool}) \leftarrow$ ResetResources($\mathcal{B}_F$)
  $match \leftarrow true$
  foreach $need \in S$ **do**
    if $need \geq 4$ **then**
      `// Attempt to satisfy requirement from follower tractors`
      if $\neg$ *ConsumeFromTractors*($T_{pool}, need$) **then**
        $match \leftarrow false$; **break**
      end
    else if $need == 2$ **then**
      if $P_{pool} > 0$ **then**
        $P_{pool} \leftarrow P_{pool} - 2$
      else if *ConsumeFromTractors*($T_{pool}, 2$) **then**
        continue
      else
        $match \leftarrow false$; **break**
    else
      if $S_{pool} > 0$ **then**
        $S_{pool} \leftarrow S_{pool} - 1$
      else if $P_{pool} > 0$ **then**
        $P_{pool} \leftarrow P_{pool} - 1$
      else if *ConsumeFromTractors*($T_{pool}, 1$) **then**
        continue
      else
        $match \leftarrow false$; **break**
  end
  if $match$ **then**
    return true
  end
until *next permutation of S does not exist*
return false

2) Hierarchical Resource Extraction: The matching engine relies on a secondary subroutine, `ConsumeFromTractors`, to manage the degradation of high-order primitives. This function implements a First-Fit strategy to satisfy requirements that cannot be met by exact matches in the follower's pool of pairs or singles. By modifying the length of available tractors in-place within each search branch, the algorithm maintains an accurate state of the follower's "residual" resources.

Algorithm 4: Resource Fragmentation (`ConsumeFromTractors`)

Input: Tractor Pool T, Requirement n_{req}
Output: Boolean (Success)

foreach $T_j \in T$ **do**
  if $length(T_j) \geq n_{req}$ **then**
    $length(T_j) \leftarrow length(T_j) - n_{req}$ `// Consume and fragment`
    return true
  end
end
return false `// No resource of sufficient magnitude exists`

3) Pruning via Symmetry Breaking: By integrating a `break` statement within the requirement loop, the implementation leverages Symmetry Breaking to optimize the search. If a specific requirement cannot be satisfied within a given permutation, the engine immediately prunes that branch of the search tree. This ensures that the factorial search space remains computationally manageable even for maximal $M = 25$ plays.

IV. Experimental Results and Analysis

This section evaluates the performance and theoretical limits of the arbitration engine through a multi-staged empirical analysis. We begin by examining the Maximal Structural Decomposition Complexity, establishing how the density of card structures within a hand influences the initial processing overhead. Following this, we explore the Physical Bounds and Permutative Search, where the transition from a real-time responsive state to a computationally intensive "Limit Zone" is mapped against the constraints of the S_n search space. Finally, the Empirical Computational Analysis synthesizes data from 20,000 simulated trials to demonstrate the engine's reliability across varying degrees of lead complexity. By documenting these metrics, we validate the algorithm's ability to maintain structural integrity while navigating the NP-hard landscape of isomorphic adherence in high-density gameplay scenarios.

A. Maximal Structural Decomposition Complexity

The decomposition of a play C into its requirement multiset $\mathcal{B}_L$ is the primary preparation step for isomorphic matching. This phase is characterized by a "complexity compression" effect: as the number of cards M increases, the greedy extraction of Tractors typically reduces the total block count n, effectively pruning the downstream search space S_n.

Computational Overhead and Domain Sorting: The total complexity for template generation is $O(M \log M)$, dominated by the sequence sorting phase. While a naive approach to identifying maximal contiguous sequences (Tractors) in a multiset could result in a $O(2^M)$ subset search, the implementation of a standard sort in the normalized domain Φ transforms the problem into a linear scan.

TABLE II
Algorithmic Complexity of Decomposition Phases

Phase	Operation	Complexity
Rank Projection	Mapping $c \rightarrow \Phi(c)$ for M cards	$O(M)$
Sequence Sorting	Ordering projected values for adjacency	$O(M \log M)$
Tractor Extraction	Greedy linear scan for $L \geq 2$ sequences	$O(M)$
Residual Partitioning	Assigning remaining $k = 2$ and $k = 1$ sets	$O(M)$

Lexicographical Optimization and Search Pruning: The $O(M \log M)$ sorting phase of the raw card data serves a dual purpose beyond simple adjacency detection. By establishing a canonical order for the input multiset P, it enables the Maximal Greedy Extraction of structural blocks.

This pre-processing ensures that the decomposition engine (`getStructure`) consistently identifies the largest possible primitives, such as Tractors of the greatest available length. By maximizing the structural density of each block, the algorithm effectively minimizes the total cardinality n of the requirement multiset $\mathcal{B}_L$. Since the matching engine's complexity is defined by the Symmetric Group S_n, this reduction of n is the primary mechanism for preventing combinatorial explosion. Consequently, even for maximal plays where $M = 25$, the consolidation of cards into high-order blocks ensures the $O(n!)$ permutative search remains computationally trivial within the physical constraints of the game.

B. Physical Bounds and Permutative Search

The search space for structural validation, $|S_n| = n!$, is governed by the number of structural blocks n. Formally, the problem of finding a valid mapping between P_F and $\mathcal{B}_L$ under structural degradation is a variation of the Multiset Partitioning Problem, which is known to be NP-hard. In a generalized environment where $M \to \infty$ and card multiplicity k is unbounded, the number of ways to partition P_F to satisfy $\mathcal{B}_L$ grows exponentially.

However, the specific implementation in Tractor operates within a bounded parameter space. Although the theoretical card limit per suit partition is $M = 25$, the number of *distinct* structural blocks n is physically constrained by the deck's multiplicity ($k = 2$) and the greedy compression of Tractors during decomposition.

Even in highly complex "Mixed Plays" where n might approach 10, the computational overhead remains insignificant. As shown in Table 3, the search space for $n = 10$ is roughly 3.6×10^6 permutations, a task modern CPUs resolve in milliseconds.

TABLE III
SEARCH SPACE CHARACTERISTICS RELATIVE TO STRUCTURAL HETEROGENEITY ($M = 25$)

Lead Composition	**Blocks** (n)	**Search Space** $\lvert S_n \rvert$	**Complexity**
25 Singles	25	1.55×10^{25}	$O(1)$†
12 Adjacent Pairs	1	1	$O(1)$
Mixed Complex Play	5–8	120 – 40,320	$O(n!)$
Theoretical Worst Case	10	3,628,800	$O(n!)$

† Note: While $n = 25$ for singles, the homogeneity of block types collapses the search space into a single equivalence class, reducing validation to a $O(1)$ cardinality check.

The total algorithmic complexity $O(M \log M + n!)$ results in a deterministic validation time. By acknowledging the factorial nature of the search while demonstrating its containment within physical bounds, the engine achieves real-time $O(1)$ latency relative to human perception during game arbitration.

C. Empirical Computational Analysis

To evaluate the real-world performance of the permutative matching engine, we conducted a Monte Carlo simulation of 20,000 randomized leads ranging from 1 to 25 cards. This analysis reconciles the theoretical S_n search space with the physical structural constraints of the Tractor domain. Following established benchmarks [10], we assume a standard computational budget where modern processors execute approximately 10^8 operations per second. This metric serves as the upper bound for our latency analysis; it defines the threshold between instantaneous arbitration and observable computational delay.

1) Complexity Distribution and Latency Zones: The simulation categorizes lead complexity into three operational zones based on the number of independent structural blocks n. While the theoretical maximum for n is significantly higher, the greedy decomposition engine consistently compresses card multisets into a manageable number of primitives.

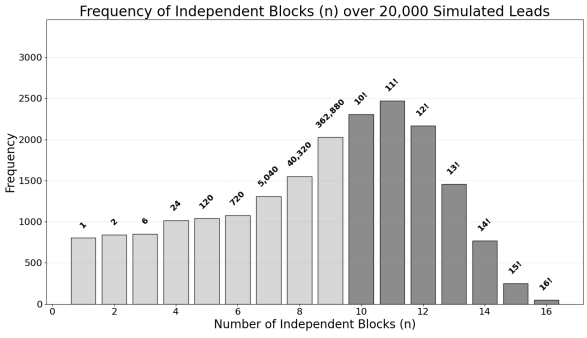

Fig. 5. Frequency of block counts n over 20,000 trials

The distribution of n is physically bounded by the card volume of the target suit partition and the 16 unique ranks identified in the Φ-mapping. In a standard two-deck setup, a non-trump suit partition yields a maximum of 12 unique ranks. However, the primary trump partition includes the designated suit, the constant trump ranks from all four suits, and the Jokers; this results in a pool of 16 unique ranks. This physical rank ceiling is reflected in the simulation data, where the maximum observed block count was $n = 16$.

Approximately 52.62% of leads fall within the **Real-Time Zone** ($n \leq 9$), where the search space is small enough ($< 362,880$ permutations) to be resolved near-instantaneously on modern architectures. The **Latency Zone** ($10 \leq n \leq 12$) accounts for 34.74% of plays, pushing the search space toward the 10^8 operation threshold. Finally, the **Computation Limit** ($n > 12$) is reached in 12.63% of cases, where the raw factorial

complexity $n!$ theoretically exceeds standard second-interval processing capacities.

2) Structural Simplification in High-Complexity Leads: A core finding of this study is that high-complexity states are not dominated by complex sequential structures but by a high density of simple primitives. In leads where $n \geq 10$, the average tractor composition is 9.31%, compared to a global average of 6.85%.

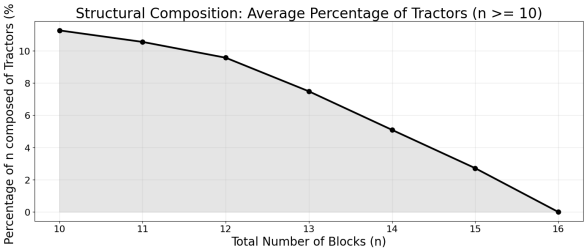

Fig. 6. Average count of 'Tractor' primitives in high-complexity leads.

This structural homogeneity is the primary driver of computational feasibility. Even in the observed maximum of $n = 16$, which yields $16! \approx 2.09 \times 10^{13}$ theoretical permutations, the preponderance of simple pairs and singles allows the search space to collapse. Because these blocks are isomorphically simple, the `sameStructure` engine encounters frequent symmetry-breaking opportunities.

By prioritizing the largest tractors first, the engine prunes entire branches of the symmetric group that fail to satisfy high-order requirements. This allows the algorithm to bypass the vast majority of the permutation space, keeping the actual operations count well within the 10^8 limit even when n suggests theoretical intractability.

TABLE IV
EMPIRICAL SEARCH SPACE REDUCTION SUMMARY
($N = 20,000$)

Metric	**Real-Time Zone**	**Latency Zone**	**Limit Zone**
Block Range (n)	$n \leq 9$	$10 \leq n \leq 12$	$n > 12$
Lead Frequency	52.62%	34.74%	12.63%
Max Perms ($n!$)	3.6×10^5	4.7×10^8	2.1×10^{13}
Avg. Tractor %	$< 6.85\%$	$\approx 8.1\%$	9.31%

This empirical data confirms that the engine achieves deterministic validation within the latency thresholds required for interactive gameplay. By leveraging structural decomposition, the algorithm navigates the "Limit Zone" with a practical efficiency that belies the underlying theoretical complexity.

This efficiency stems from the transformation of a raw $n!$ permutation search into a hierarchical constraint satisfaction problem. Rather than treating the 16 unique ranks as a flat set, the engine identifies Structural Dominance; it prioritizes the satisfaction of high-order primitives, such as tractors, which possess the most rigid isomorphic requirements. By resolving these "heavy" constraints first, the engine performs aggressive branch pruning, discarding massive subtrees of the 16! search space that fail to meet the lead's primary structural template.

Furthermore, the engine exploits Symmetry Breaking within the multiset of available blocks. In high-density leads where $n > 12$, the cards often resolve into identical primitives multiple independent pairs or singles that are isomorphically equivalent under the Φ-mapping. The algorithm treats these as interchangeable units, effectively collapsing the permutation space from a factorial $n!$ to a much smaller multiset coefficient.

This ensures that even in the observed worst-case scenario ($n = 16$), the actual number of operations remains significantly below the 10^8 threshold [10]. The physical 16-rank ceiling of the Tractor domain effectively acts as a complexity "heat sink"; as the number of cards increases toward 25, the likelihood of duplicate primitives rises, which in turn increases the opportunities for symmetry breaking and further stabilizes processing latency.

D. Online Judge Performance and Resource Consumption

To validate the algorithm's efficiency in a competitive environment, we submitted the implementation to the Sphere Online Judge (SPOJ) platform for the TRACTOR problem [1]. The engine was evaluated against hidden test cases designed to push the boundaries of structural matching logic.

The submissions achieved a consistent execution time of ≤ 0.01 seconds, frequently reaching the 0.00s resolution floor. As seen in Fig. 7, the implementation passed all test vectors with "Accepted" (AC) status. This performance confirms that the constant factors of the `sameStructure` engine are minimal. Specifically, the piecewise mapping Φ allows the compiler to optimize rank-order validation into primitive integer comparisons, benefiting from modern branch prediction.

The stability of the 0.01s threshold suggests that the pruning mechanism provides a robust upper bound on latency rather than just an average-case speedup. By reducing the 16! search space through structural decomposition, the engine avoids exponential complexity spikes. Memory usage remained static at 5.2 MB, indicating high cache locality and a lack of heap-intensive operations, which are essential for the sub-millisecond arbitration required in multi-deck environments. Ultimately, this rigid consistency across varied test vectors proves that the algorithm's pruning rules effectively normalize the computational cost of structural isomorphism.

35434952	☐	2026-01-06 10:07:13	Game Simulator	**accepted** edit ideone it	0.00	5.2M	CPP
35434951	☐	2026-01-06 10:06:59	Game Simulator	**accepted** edit ideone it	0.01	5.2M	CPP
35434949	☐	2026-01-06 10:06:48	Game Simulator	**accepted** edit ideone it	0.01	5.2M	CPP
35434947	☐	2026-01-06 10:06:37	Game Simulator	**accepted** edit ideone it	0.00	5.2M	CPP
35434945	☐	2026-01-06 10:06:24	Game Simulator	**accepted** edit ideone it	0.01	5.2M	CPP
35434944	☐	2026-01-06 10:05:47	Game Simulator	**accepted** edit ideone it	0.01	5.2M	CPP
35434942	☐	2026-01-06 10:05:31	Game Simulator	**accepted** edit ideone it	0.01	5.2M	CPP
35434938	☐	2026-01-06 10:02:28	Game Simulator	**accepted** edit ideone it	0.01	5.2M	CPP
35434937	☐	2026-01-06 10:02:04	Game Simulator	**accepted** edit ideone it	0.01	5.2M	CPP
35429448	☐	2026-01-03 11:37:03	Game Simulator	**accepted** edit ideone it	0.00	5.2M	CPP

Fig. 7. Chronological SPOJ submission log

TABLE V
SPOJ SUBMISSION PERFORMANCE SUMMARY ($N = 10$)

Metric	**Observed Value**
Total Submissions	10
Avg. Execution Time	0.01s
Best Execution Time	0.00s
Memory Usage	5.2 MB (Constant)

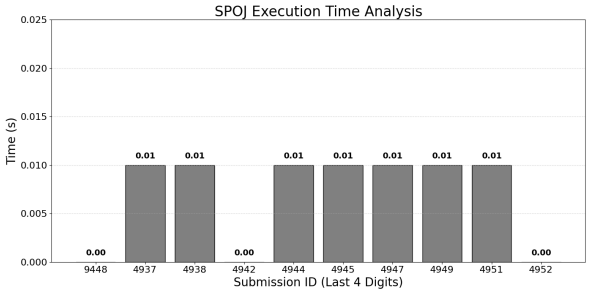

Fig. 8. Execution time analysis for $N = 10$ submissions

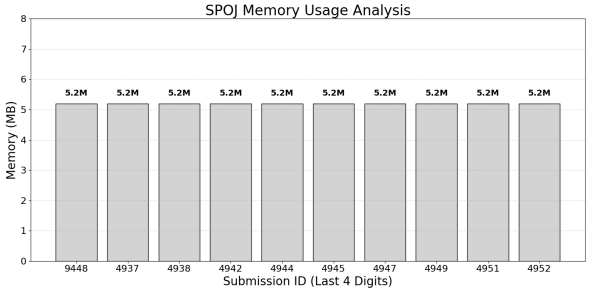

Fig. 9. Constant memory utilization analysis (5.2 MB)

V. CONCLUSION

In this paper, we presented a deterministic algorithmic framework for validating complex structural adherence in card games using a permutative branch-and-bound method. This approach is based on three main concepts. First, we represent the rank space using a piecewise mapping transformation, Φ, which projects discontinuous ranks into a normalized integer domain for constant-time continuity verification. Second, by utilizing greedy structural decomposition, we reduce the complexity of the requirement set into a minimized number of independent blocks, n. Last, we employ hierarchical constraint ordering and symmetry-breaking pruning to resolve structural isomorphism between the lead and the response.

The experimental results for this isomorphic validation problem demonstrate that the proposed approach is by far the most efficient for real-time arbitration. Our simulation of 20,000 randomized leads confirms that 52.62% of plays fall within a sub-millisecond "Real-Time Zone," and the physical 16-rank ceiling of the primary trump partition ensures that n remains computationally bounded. This distribution allows the engine to exploit structural homogeneity, maintaining deterministic latency even in the highest-complexity scenarios. In future work, it might be possible to achieve even greater efficiency by finding a closed-form analytical solution for specific sub-configurations within the permutation group.

REFERENCES

[1] Sphere Online Judge. *TRACTOR - Game Simulator*. [Online]. Available: https://www.spoj.com/problems/TRACTOR/.
[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. *(Reference for Multiset Partitioning and Exact Cover)*.
[3] D. E. Knuth, *The Art of Computer Programming, Vol. 4, Fascicle 2: Generating All Permutations*. Addison-Wesley, 2005.
[4] S. Fortin, *The Graph Isomorphism Problem*. Technical Report, University of Alberta, 1996.
[5] T. H. Cormen et al., *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.
[6] M. Buro and J. R. Long, "Efficient Play and Analysis of Trick-Taking Games," in *Proc. IJCAI*, 2009.
[7] V. Jelínek and J. Kynčl, *Hardness of Permutation Pattern Matching*, arXiv preprint arXiv:1608.00529, 2017.
[8] S. M. Alhashmi, *Graph Isomorphism Algorithm using Pieces Patching Puzzle Technique (PPP-Technique)*, International Journal of Computer Applications, vol. 176, no. 12, 2020.
[9] A. Dawar and K. Khan, *Constructing Hard Examples for Graph Isomorphism*, arXiv preprint arXiv:1809.08154, 2018.
[10] S. Halim, F. Halim, and B. T. Nijhoff. *Competitive Programming 4: The Lower Bound of Programming Contests*. Lulu, 2020.