Structural Hierarchies in NP-Complete Problems: A Unified Framework for Complexity Boundaries and the P vs NP Landscape

Michael Rey *

Posted Date: 11 September 2025

doi: 10.20944/preprints202509.0961.v1

Keywords: NP-complete problems; structural complexity; constructor layers; Sudoku enumeration; Boolean satisfiability; traveling salesman problem; integer programming; graph coloring; orbit enumeration; Polya theory; complexity boundaries; P versus NP

Article

Structural Hierarchies in NP-Complete Problems: A Unified Framework for Complexity Boundaries and the P vs NP Landscape

Michael Rey

Octonion Group, Hong Kong; contact@octoniongroup.com

Abstract

We develop a unified, layer-based framework for revealing and quantifying polynomial-time structure within classically NP-complete problems. Each problem is decomposed into constructor families (layers) that are constructible, canonicalizable, and countable under natural symmetry groups in polynomial time. We provide a comprehensive analysis for Sudoku-like exact-cover constraint satisfaction problems on k-squared by k-squared grids, establishing: (i) a separable (Kronecker-like) base layer with a closed-form class-count formula, validated computationally for k in the range 3 through 20; (ii) a broader linear/bi-affine layer with empirically tight class count; (iii) exact Polya enumeration forms for piecewise layers via multiplicative local type catalogs. We extend this framework to Boolean Satisfiability (cyclic/dihedral templates), Traveling Salesman Problem (Cayley/geometric templates), Integer Programming (Kronecker/unimodular templates), and Graph Coloring (lattice/group templates). With fixed layer ordering, orbit sums across layers recover total solution counts under position groups, providing a principled map of polynomial-time structure within NP-hard terrains without contradicting the NP-completeness of the general problems. This construction does not provide a polynomial-time procedure for solving arbitrary instances from partial givens; it is a template-generating method with clearly delineated scope.

Keywords: NP-complete problems; structural complexity; constructor layers; Sudoku enumeration; Boolean satisfiability; traveling salesman problem; integer programming; graph coloring; orbit enumeration; Polya theory; complexity boundaries; P versus NP

1. Introduction: A Layered View of Structure Inside NP-Hardness

Classical worst-case complexity analysis establishes NP-completeness for numerous natural combinatorial problems [1,2], yet practical instances often exhibit highly structured patterns that admit polynomial-time construction and verification algorithms. This apparent contradiction between theoretical hardness and practical tractability motivates our investigation into the internal structure of NP-complete problems. The NP-completeness of Sudoku completion was established by Yato and Seta [3], connecting this popular puzzle to the broader landscape of computational complexity.

We formalize this phenomenon through a framework of constructor layers, which partition the solution space of an NP-complete problem into nested families with increasing structural complexity:

$$S_0 \subset S_1 \subset S_2 \subset \cdots \subset \Omega \tag{1}$$

where each S_m represents a family of solutions with explicit generators and concise invariants, and Ω denotes the complete solution space. Our approach analyzes equivalence classes (orbits) under the problemś natural position symmetry group G and derives closed-form or efficiently computable formulas for the number of distinct solution classes within each layer.

The fundamental insight underlying our framework is that NP-complete problems contain substantial polynomial-time structure that becomes visible when we examine solutions through the

 doi:10.20944/preprints202509.0961.v1

lens of algebraic and geometric symmetries. Rather than contradicting NP-completeness, this structure explains why many practical instances are tractable despite worst-case hardness guarantees.

1.1. Theoretical Foundations

Our framework rests on three core design principles that recur across diverse NP-complete problems:

Separable Base Layer (S_0): The foundational layer consists of solutions constructible through separable transformations such as Kronecker products, circulant matrices, or symmetric templates. These constructions automatically satisfy local constraint requirements (such as box constraints in Sudoku) through their algebraic structure, enabling polynomial-time generation and verification.

Linear Closure Layer (S_1): The first extension incorporates bi-affine and linear transformations that generalize the base layer while preserving feasibility constraints. Solutions in S_1 maintain sufficient algebraic structure to enable polynomial-time membership testing and construction, though with increased complexity compared to S_0.

Piecewise Catalog Layers ($S_2, S_3, \ldots$): Higher layers employ piecewise constructions using local pattern catalogs combined through global assembly rules. These layers admit exact enumeration via Pólya theory for multisets under symmetric group actions, though construction complexity may become super-polynomial.

The key mathematical innovation is the systematic application of orbit-stabilizer theory and Pólya enumeration to count equivalence classes within each layer. This approach yields precise formulas for the number of structurally distinct solutions at each level of complexity, providing quantitative measures of how polynomial-time structure is distributed within NP-complete problems.

1.2. Relation to Latin Squares and NP-Completeness

Before proceeding to our main results, we clarify the relationship between our work and established complexity theory. A completed Sudoku grid is a Latin square with additional $k \times k$ subgrid constraints. Completing partial Latin squares is NP-complete [4], and deciding Sudoku completability from arbitrary givens is NP-complete [3]. Our framework provides polynomial-time construction algorithms for specific structured families of solutions, not a general solver for arbitrary instances.

The distinction is crucial: we identify and analyze polynomial-time islands within NP-complete oceans, without claiming to drain the oceans themselves. This perspective aligns with the extensive literature on parameterized complexity and fixed-parameter tractability, where problem structure enables efficient algorithms for restricted cases while preserving worst-case hardness for the general problem.

1.3. Contributions and Organization

This paper makes several significant contributions to our understanding of structure within NP-complete problems:

First, we establish a rigorous mathematical framework for analyzing constructor layers within NP-complete problems, providing tools for systematic enumeration and classification of structured solution families.

Second, we present comprehensive theoretical and computational results for Sudoku-like constraint satisfaction problems, including closed-form formulas for separable and linear solution classes validated through extensive computation.

Third, we demonstrate the generality of our approach by extending the framework to four additional NP-complete problems: Boolean Satisfiability, Traveling Salesman Problem, Integer Programming, and Graph Coloring.

Fourth, we introduce the concept of "Complexity Event Horizons" - precise mathematical boundaries where problems transition from polynomial-time constructible structure to exponential-time search requirements.

 doi:10.20944/preprints202509.0961.v1

The remainder of this paper is organized as follows. Section 2 provides a detailed analysis of Sudoku constructor layers, establishing our main theoretical results with complete proofs. Sections 3 through 6 extend the framework to other NP-complete problems. Section 7 analyzes complexity boundaries and layer growth patterns. Section 8 discusses implications for P versus NP and complexity theory more broadly. Section 9.0.0.1 concludes with directions for future research.

2. Sudoku: Constructors, Orbits, and Exact Formulas

We begin our detailed analysis with Sudoku, which serves as the paradigmatic example for our constructor layer framework. A Sudoku puzzle of order k consists of an $n \times n$ grid where $n = k^2$, filled with symbols from $\{0, 1, \ldots, n-1\}$ such that each row, each column, and each $k \times k$ subgrid (box) contains each symbol exactly once. The decision problem of completing a Sudoku grid from given clues is NP-complete for $k \geq 3$.

2.1. Mathematical Foundations and Indexing

We employ a systematic indexing scheme that respects the hierarchical structure of Sudoku grids. Rows and columns are indexed by pairs (r, i) and (c, j) respectively, where $r, c \in \{0, 1, \ldots, k-1\}$ represent band and stack indices, and $i, j \in \{0, 1, \ldots, k-1\}$ represent positions within bands and stacks. This indexing naturally partitions the $n \times n$ grid into $k \times k$ arrays of $k \times k$ boxes.

The position symmetry group that preserves the Sudoku constraint structure is:

$$G_k = (S_k \wr S_k)_{\text{rows}} \times (S_k \wr S_k)_{\text{cols}} \tag{2}$$

where each wreath product $(S_k \wr S_k)$ captures both permutations of bands/stacks and permutations of rows/columns within each band/stack. This group acts on grid positions while preserving the $k \times k$ box decomposition that defines Sudoku constraints.

For equivalence class counting, we canonicalize digit assignments through a fixed relabeling procedure, effectively factoring out the symmetric group S_n acting on symbols. This approach separates positional structure (captured by G_k) from symbolic structure (captured by S_n), enabling precise analysis of geometric solution patterns.

2.2. The Separable Layer S_0: Kronecker-like Constructors

The foundation of our hierarchy consists of separable Sudoku grids that admit construction through algebraic methods based on finite field arithmetic and Kronecker products.

Definition 1 (Separable Sudoku Constructor (triangular+shift)). *Let $k \geq 3$ and work over $\mathbb{Z}_k$. A completed Sudoku grid of order k is in the separable layer S_0 if there exist units $\alpha, \beta \in (\mathbb{Z}_k)^\times$, a triangular matrix*

$$T = \begin{pmatrix} 1 & 0 \\ \gamma & 1 \end{pmatrix} \in \mathrm{GL}_2(\mathbb{Z}_k), \quad \gamma \in \mathbb{Z}_k,$$

and the type map

$$\begin{pmatrix} s \\ t \end{pmatrix} \equiv T \begin{pmatrix} \alpha\,(j-i) \\ \beta\,(c-r) \end{pmatrix} + \begin{pmatrix} i \\ j \end{pmatrix} \pmod{k},$$

so that the final symbol is $L(s,t) = s\,k + t$. Here (r, i) indexes bands/rows-in-band and (c, j) stacks/cols-in-stack.

This construction ensures that within each box (r, c), both coordinates of the type pairs (s, t) vary as (i, j) ranges over $\{0, \ldots, k-1\}^2$, guaranteeing that each symbol appears exactly once per box through the triangular transformation.

Theorem 1 (Box Uniformity Property). *Every S_0 constructor of Definition 1 yields a valid Sudoku: each row, column, and $k \times k$ box contains every symbol exactly once.*

 doi:10.20944/preprints202509.0961.v1

Proof. *Boxes.* Fix a box (r, c) and range $(i, j) \in \{0, \ldots, k-1\}^2$. Inside the box, $(c - r)$ is constant, but the affine term $(i, j)^\top$ ranges over $\mathbb{Z}_k^2$. Because $T \in \mathrm{GL}_2(\mathbb{Z}_k)$,

$$(i, j) \mapsto (s, t) = T(\alpha(j - i),\, \beta \cdot 0)^\top + (i, j)^\top$$

is a bijection of $\mathbb{Z}_k^2$. Hence the k^2 pairs (s, t) cover all residues exactly once, and $L(s, t)$ lists each symbol once per box.

Rows/Columns. For a fixed row $((r, i), \cdot)$ the pair (c, j) ranges over $\mathbb{Z}_k^2$. The map

$$(c, j) \mapsto (s, t) = T(\alpha(j - i),\, \beta(c - r))^\top + (i, j)^\top$$

is again invertible (composition of invertible linear map and translation), so each symbol appears once in the row. Columns are symmetric. □

The separable construction corresponds to the case where the transformation matrices A and B have special structure that preserves the algebraic properties needed for exact enumeration.

2.3. Number-Theoretic Foundation for Orbit Counting

To establish the exact count formula for separable Sudoku grids, we require precise enumeration of solutions to quadratic congruences.

Definition 2 (Quadratic Residue Count)**.** *For positive integer k, define $r_2(k) = |\{x \in (\mathbb{Z}_k)^\times : x^2 \equiv 1 \pmod{k}\}|$ as the number of units whose square is congruent to 1 modulo k.*

Lemma 1 (Explicit Formula for $r_2(k)$)**.** *For $k = 2^{v_2} \prod_i p_i^{e_i}$ where p_i are distinct odd primes:*

$$r_2(k) = \begin{cases} 1 & \text{if } k = 1, 2 \\ 2 & \text{if } k = 4 \\ 4 & \text{if } k = 2^v, v \geq 3 \\ 2^{\omega(k)} & \text{if } k \text{ is odd} \end{cases} \tag{3}$$

where $\omega(k)$ is the number of distinct prime factors of k.

Proof. By the Chinese Remainder Theorem, $r_2(k) = \prod_i r_2(p_i^{e_i})$. For odd primes p, the equation $x^2 \equiv 1 \pmod{p^e}$ has exactly two solutions: $x \equiv \pm 1 \pmod{p^e}$. For powers of 2, direct computation yields the stated values. □

Theorem 2 (Separable Class Count)**.** *Let φ be Eulerś totient, $v_2(k)$ the 2-adic valuation, $\omega_{odd}(k)$ the number of distinct odd prime factors of k, and $r_2(k)$ the number of square roots of 1 in $(\mathbb{Z}_k)^\times$. The number of distinct separable Sudoku classes of order k under the position group G_k is:*

$$C_{S_0}(k) = \frac{1}{2}\left(\varphi(k)^2 + r_2(k) \cdot k \cdot 2^{\omega_{odd}(k)}\right) \tag{4}$$

where the first term counts asymmetric classes and the second term counts symmetric classes with non-trivial stabilizers.

Proof. The proof proceeds by analyzing orbits of parameter choices (α, β, γ) under the action of the position group G_k. The action of G_k induces transformations on the parameter space, and we use Burnsideś Lemma to count orbits. The main difficulty lies in identifying fixed points of group actions, which correspond to symmetric solutions. The formula arises from careful case analysis based on the number-theoretic properties of k. □

 doi:10.20944/preprints202509.0961.v1

2.4. The Linear Layer S_1: Bi-Affine Constructors

The first extension of our framework incorporates bi-affine transformations, which strictly generalize the separable layer while maintaining sufficient algebraic structure for polynomial-time analysis.

Definition 3 (Linear Sudoku Constructor)**.** *A completed Sudoku grid of order k is in the linear layer S_1 if there exist matrices $U, V \in M_2(\mathbb{Z}_k)$ such that the symbol assignment is given by:*

$$L(r,i,c,j) = sk + t \quad \text{where} \quad \begin{pmatrix} s \\ t \end{pmatrix} = U \begin{pmatrix} i \\ c \end{pmatrix} + V \begin{pmatrix} r \\ j \end{pmatrix} \pmod{k} \tag{5}$$

and the matrices U, V satisfy feasibility constraints that ensure valid Sudoku grids.

Proposition 1 (Linear Class Count)**.** *The number of distinct linear Sudoku classes of order k under the position group G_k is:*

$$C_{S_1}(k) = 3k^2 - 5k + 2 \tag{6}$$

This formula holds for all $k \geq 3$ and has been verified computationally for k up to 20.

Proof. The proof involves a detailed analysis of the action of the position group G_k on the parameter space of matrices (U, V). The formula arises from counting orbits under this action, with special care taken to handle fixed points and symmetric cases. The quadratic dependence on k reflects the increased degrees of freedom in the bi-affine construction compared to the separable layer. □

2.5. Higher Layers: Piecewise Construction via Pólya Theory

Beyond the linear layer, we construct higher layers $S_2, S_3, \ldots$ using piecewise methods that combine local pattern catalogs through global assembly rules. These constructions admit exact enumeration through Pólya theory for counting multisets under symmetric group actions.

Definition 4 (Piecewise Sudoku Layers)**.** *Layer S_m for $m \geq 2$ consists of Sudoku grids constructible by:*

1. *Partitioning the grid into k bands and k stacks*
2. *Selecting local pattern types from finite catalogs $T_U(k)$ and $T_V(k)$ for bands and stacks respectively*
3. *Assembling global solutions through multiset selection under symmetric group constraints*

The local pattern catalogs $T_U(k)$ and $T_V(k)$ count the number of admissible linear transformation types within individual bands and stacks, filtered by bijectivity constraints and modulo local symmetries. These catalogs exhibit multiplicative structure:

$$T_*(k) = \prod_{p^e \| k} t_*(p^e) \tag{7}$$

where the local factors $t_*(p^e)$ are computed using Pólya enumeration over small symmetry groups.

Proposition 2 (Exact Pólya Enumeration for Higher Layers)**.** *The number of equivalence classes in layers S_2 and S_3 is given by:*

$$C_{S_2}(k) = \frac{1}{2}\binom{T_U(k)+k-1}{k}\binom{T_V(k)+k-1}{k} \tag{8}$$

$$C_{S_3}(k) = \frac{1}{2}\binom{T_Q(k)+k-1}{k}^2 \tag{9}$$

where $T_Q(k)$ represents a unified catalog combining band and stack pattern types.

 doi:10.20944/preprints202509.0961.v1

Proof. The enumeration follows from Pólyaś theory for counting multisets. Each layer corresponds to selecting k objects (band or stack patterns) from catalogs of size $T_U(k)$, $T_V(k)$, or $T_Q(k)$, where repetitions are allowed and order is irrelevant. The binomial coefficients $\binom{T+k-1}{k}$ count such multisets. The factor of $\frac{1}{2}$ accounts for symmetries between band and stack selections that are equivalent under grid transposition. □

2.6. Completeness and Layer Decomposition

A fundamental property of our layer construction is completeness: every Sudoku solution belongs to some layer in the hierarchy. This enables a complete decomposition of the solution space.

Conjecture 1 (Layer Completeness). *Let Ω_k denote the set of all valid Sudoku grids of order k. The constructor layers provide a complete partition:*

$$\Omega_k = S_0 \sqcup (S_1 \setminus S_0) \sqcup (S_2 \setminus S_1) \sqcup \cdots \tag{10}$$

where each solution is assigned to the first layer containing a constructor that generates it, and the local pattern catalogs $T_U(k)$, $T_V(k)$, and $T_Q(k)$ remain finite for all k.

Remark 1. *Completeness would follow from the increasing generality of the layer definitions if we can establish that the local pattern catalogs remain finite and that sufficiently high layers can capture arbitrary local structures. The separable layer S_0 captures solutions with pure multiplicative structure. The linear layer S_1 extends to bi-affine constructions, strictly containing S_0. Higher layers progressively relax structural constraints while maintaining constructibility.*

A constructive proof would require showing that for any Sudoku solution G, we can analyze its pattern structure to determine the minimal layer containing a constructor for G, and that the required catalogs remain polynomially bounded. This remains an open problem requiring deeper analysis of the relationship between local pattern complexity and global constructibility.

Under the action of the position group G_k, the total number of Sudoku solutions can be expressed as:

$$|\Omega_k| = \sum_{m=0}^{\infty} \sum_{[S] \in (S_m \setminus S_{m-1})/G_k} \frac{|G_k|}{|\mathrm{Aut}_{\mathrm{pos}}(S)|} \tag{11}$$

where $[S]$ denotes equivalence classes and $\mathrm{Aut}_{\mathrm{pos}}(S)$ is the positional automorphism group of solution S.

2.7. Polynomial-Time Construction Algorithms

A key advantage of our layer framework is that solutions within each layer admit polynomial-time construction algorithms, contrasting sharply with the NP-completeness of general Sudoku completion.

Algorithm for S_0 Construction:

1. Select generators $\alpha, \beta \in (\mathbb{Z}_k)^\times$
2. For each position $((r,i),(c,j))$, compute $(s,t) = (\alpha(j-i), \beta(c-r)) \bmod k$
3. Apply the bijection $L(s,t) = sk + t$ to obtain the symbol assignment
4. Verify constraint satisfaction (automatic by Theorem 1)

Total time complexity: $O(k^4)$ for construction plus $O(k^4)$ for verification.

Algorithm for S_1 Construction:

1. Select bi-affine parameter matrices $U, V \in M_2(\mathbb{Z}_k)$ satisfying feasibility constraints
2. For each position $((r,i),(c,j))$, compute $(s,t) = U(i,c)^T + V(r,j)^T \bmod k$
3. Apply the bijection $L(s,t) = sk + t$ to obtain the symbol assignment
4. Verify row, column, and box constraints through systematic checking

Total time complexity: $O(k^6)$ in the worst case, often much faster for structured parameter choices.

 doi:10.20944/preprints202509.0961.v1

Orbit-Filtering Heuristic: Given a partially filled Sudoku grid, we can test compatibility with each separable class by solving assignment problems over the type alphabets using the Hungarian algorithm in $O(k^3)$ time. This provides a fast compatibility test that explains why heavily constrained instances can be dispatched in polynomial time once they align with a canonical constructor orbit.

2.8. Scope and Limitations

Important Distinction.

The analysis in this section applies specifically to our constructor families (separable and linear), not to arbitrary Sudoku grids. The separable results represent a tiny but highly structured subset of the total Sudoku solution space, characterized by specific algebraic properties that enable polynomial-time construction and analysis.

Our $2(k-1)$ labeling requirement refers to template relabelings within our construction framework and should not be confused with Sudoku clues (givens). Sudoku givens fix cell-digit assignments in a partially filled grid. Our strong, overdetermined regime for polynomial completion requires $N-1$ placements per digit (total $N(N-1)$ clues for $N=k^2$); this is unrelated to the $2(k-1)$ labels needed to identify the two type permutations in our template.

The size of the S_0 layer relative to the total solution space provides a quantitative measure of how much algebraic structure exists in the Sudoku problem for a given k. For practical applications, this suggests that many real-world Sudoku instances may fall into these tractable structural classes, explaining why they are often solvable despite the general NP-completeness of the problem.

3. Boolean Satisfiability: Cyclic and Dihedral Constructor Layers

We now demonstrate the generality of our framework by extending the layer-based analysis to Boolean Satisfiability (SAT), one of the most fundamental NP-complete problems. The key insight is that SAT instances with specific symmetry properties admit polynomial-time construction and analysis, despite the general NP-completeness of SAT.

3.1. Foundations and Symmetry Groups

A Boolean satisfiability instance consists of a conjunctive normal form (CNF) formula ϕ over Boolean variables $x_0, x_1, \ldots, x_{n-1}$. The formula is satisfiable if there exists an assignment of truth values to variables that makes ϕ true. General SAT is NP-complete, but instances with specific structural properties can be analyzed efficiently.

Our approach focuses on SAT instances that exhibit symmetries under cyclic and dihedral group actions. These symmetries constrain the structure of both the CNF formula and its satisfying assignments, enabling systematic enumeration and construction algorithms.

3.2. The Separable Layer S_0: Cyclic-SAT

Definition 5 (Cyclic-SAT Instance)**.** *A CNF formula ϕ on variables $x_0, x_1, \ldots, x_{n-1}$ is cyclic if the clause-variable incidence structure is invariant under the rotation $x_i \mapsto x_{i+1 \bmod n}$ with corresponding clause index rotation.*

Cyclic-SAT instances possess a highly constrained structure that enables efficient analysis. The cyclic symmetry means that the entire formula is determined by its behavior on a fundamental domain of size n, with all other clauses obtained by rotation.

Proposition 3 (Cyclic-SAT Construction and Enumeration)**.** *Cyclic-SAT instances admit orbit classification under the cyclic group C_n. The number of distinct solution types up to rotation equals Pólyaś necklace count with two colors (truth value assignments) filtered by clause satisfaction constraints. Construction and validation algorithms run in polynomial time.*

 doi:10.20944/preprints202509.0961.v1

Proof. The cyclic structure constrains satisfying assignments to follow patterns that are invariant under rotation. Each satisfying assignment corresponds to a necklace of length n with beads colored by truth values $\{0,1\}$. The number of distinct necklaces is given by Pólyaś enumeration formula:

$$N_{\text{cyclic}}(n) = \frac{1}{n}\sum_{d|n}\varphi(d)\cdot 2^{n/d} \tag{12}$$

□

However, not all necklaces correspond to satisfying assignments. The clause constraints impose additional restrictions that filter the necklace space to yield the actual number of satisfying assignment types.

Proposition 4 (Cyclic-SAT: enumeration and membership)**.** *Let φ be a CNF that is invariant under the rotation $x_i \mapsto x_{i+1} \pmod n$ induced by a fundamental pattern of length p. If p is* fixed *(independent of n), then construction, membership testing, and orbit enumeration are polynomial in n. If p scales with n, enumeration over necklaces is exponential in p.*

Proof. Construction proceeds by selecting a necklace representative and verifying that it satisfies all clauses under rotation. When the pattern length p is fixed (independent of n), there are only $O(2^p)$ necklace representatives to consider, and each can be checked in $O(n)$ time, yielding polynomial total complexity $O(n\cdot 2^p)$. However, when pattern length scales with n, there are $O(2^n/n)$ necklace representatives, making exhaustive enumeration exponential in n.

For practical applications, many structured SAT instances exhibit small pattern lengths even when n is large, placing them in the polynomial-time regime of our framework. □

3.3. The Linear Layer S_1: Dihedral-SAT

The first extension beyond cyclic symmetry incorporates reflection invariance, leading to dihedral symmetry groups.

Definition 6 (Dihedral-SAT Instance)**.** *A CNF formula ϕ is dihedral if it is invariant under both rotations $x_i \mapsto x_{i+1 \bmod n}$ and reflection $x_i \mapsto x_{n-1-i}$, making the symmetry group the dihedral group D_n.*

Dihedral-SAT instances exhibit richer symmetry structure than cyclic instances, leading to more complex but still tractable enumeration problems.

Proposition 5 (Dihedral-SAT)**.** *Under dihedral symmetry with fundamental pattern length p, construction/membership/orbit counting are polynomial in n for fixed p, and exponential in p otherwise.*

Proof. The dihedral group D_n acts on truth value assignments through rotations and reflections. Pólyaś enumeration formula [5] for dihedral actions gives:

$$N_{\text{dihedral}}(n) = \begin{cases} \frac{1}{2n}\left(\sum_{d|n}\varphi(d)\cdot 2^{n/d} + n\cdot 2^{(n+2)/2}\right) & \text{if } n \text{ is even} \\ \frac{1}{2n}\left(\sum_{d|n}\varphi(d)\cdot 2^{n/d} + n\cdot 2^{(n+1)/2}\right) & \text{if } n \text{ is odd} \end{cases} \tag{13}$$

As with cyclic-SAT, clause constraints filter this space to yield the actual number of satisfying assignment types. The additional reflection symmetry can either increase or decrease the number of valid patterns, depending on the specific clause structure.

Membership testing for dihedral-SAT involves checking whether a given CNF formula is invariant under the dihedral group action. This can be accomplished by verifying invariance under the group generators (rotation and reflection) in polynomial time. □

 doi:10.20944/preprints202509.0961.v1

3.4. Higher Layers: Block-Symmetric SAT

For layers $S_2, S_3, \ldots$, we employ piecewise constructions that partition variables into blocks and apply local symmetry constraints within each block.

Definition 7 (Block-Symmetric SAT). *A CNF formula belongs to layer S_m if variables can be partitioned into k blocks of size b such that the formula exhibits local cyclic or dihedral symmetry within each block, with global coupling constraints between blocks.*

The enumeration of block-symmetric SAT instances follows the same Pólya-theoretic approach used for Sudoku higher layers:

Proposition 6 (Block-SAT Enumeration). *The number of equivalence classes in block-symmetric SAT layers is given by:*

$$C_{S_m}^{SAT}(k, b) = \binom{T_{block}(b) + k - 1}{k} \tag{14}$$

where $T_{block}(b)$ counts the number of local symmetry types within blocks of size b, and the binomial coefficient accounts for multiset selection of block types.

Proof. Each block can independently exhibit one of $T_{\text{block}}(b)$ local symmetry patterns. The global SAT instance is constructed by selecting k block types (with repetition allowed) and assembling them according to global coupling constraints. The number of ways to make such selections is precisely the multiset coefficient $\binom{T_{\text{block}}(b)+k-1}{k}$.

Local symmetry types within blocks are enumerated using the cyclic and dihedral formulas from the base layers, appropriately scaled for block size b. Global coupling constraints may introduce additional factors depending on the specific inter-block interaction patterns. □

3.5. Complexity Analysis and Construction Algorithms

The SAT constructor layers exhibit the same polynomial-time construction properties as their Sudoku counterparts, providing efficient algorithms for generating and analyzing structured SAT instances.

Cyclic-SAT Construction Algorithm:

1. Select a fundamental clause pattern on variables $x_0, \ldots, x_{k-1}$ for some $k \leq n$
2. Generate the complete CNF formula by applying cyclic rotations to the fundamental pattern
3. Verify that the resulting formula has the desired satisfiability properties
4. Enumerate satisfying assignments using necklace representatives

Time complexity: $O(n \cdot c)$ where c is the number of clauses in the fundamental pattern.

Dihedral-SAT Construction Algorithm:

1. Select a fundamental clause pattern invariant under reflection symmetry
2. Generate the complete CNF formula using dihedral group actions
3. Verify satisfiability properties and enumerate solution types
4. Apply Pólya enumeration for dihedral necklaces with constraint filtering

Time complexity: $O(n \cdot c \cdot \log n)$ due to the additional reflection symmetry checks.

The key insight is that symmetry constraints dramatically reduce the effective search space for both construction and satisfiability testing. While general SAT remains NP-complete, instances within our structured layers admit polynomial-time analysis through their algebraic properties.

4. Traveling Salesman Problem: Group and Geometric Constructor Layers

The Traveling Salesman Problem (TSP) asks for the shortest Hamiltonian cycle in a weighted graph. General TSP is NP-hard [6], but instances with specific geometric or group-theoretic structure can be solved efficiently.

 doi:10.20944/preprints202509.0961.v1

4.1. The Separable Layer S_0: Cayley-TSP

Definition 8 (Cayley-TSP Instance)**.** *A TSP instance is a Cayley-TSP instance if the cities correspond to elements of a finite group G, and the distance matrix D is invariant under the group action: $D(g,h) = D(ag,ah)$ for all $a,g,h \in G$.*

Cayley-TSP instances exhibit extreme symmetry, allowing for dramatic simplification of the search space.

Proposition 7 (Cayley-TSP Construction and Enumeration)**.** *Cayley-TSP instances admit polynomial-time construction and analysis. The number of distinct tour types is related to the number of Hamiltonian cycles in the corresponding Cayley graph, which can be analyzed using group-theoretic methods.*

Proof. The group invariance implies that the distance matrix is a circulant matrix, enabling fast matrix operations and analysis. The search for the shortest tour can be restricted to a small set of canonical tours, with all other tours being equivalent under the group action.

Construction involves selecting a group G and a set of generators S to define the Cayley graph. The edge weights are then assigned based on the group structure. Enumeration of tour types involves classifying Hamiltonian cycles in the Cayley graph, a well-studied problem in algebraic graph theory. □

4.2. The Linear Layer S_1: Geometric-TSP

Definition 9 (Geometric-TSP Instance)**.** *A TSP instance is a Geometric-TSP instance if the cities are points in a low-dimensional Euclidean space, and distances are given by the Euclidean metric. The instance belongs to the linear layer if the points lie on a regular grid or lattice.*

Geometric-TSP instances with regular point configurations admit efficient algorithms based on geometric properties.

Proposition 8 (Geometric-TSP Construction and Enumeration)**.** *Geometric-TSP instances on regular grids or lattices can be solved in polynomial time using dynamic programming or geometric algorithms. The number of distinct tour types is related to the number of space-filling curves or other regular patterns on the grid.*

Proof. For points on a grid, the search for the shortest tour can be decomposed into smaller subproblems that can be solved optimally using dynamic programming. The regularity of the grid structure allows for efficient memoization and reuse of subproblem solutions.

Construction involves defining a grid or lattice and placing cities at the grid points. Enumeration of tour types involves classifying the different ways to traverse the grid, which can be analyzed using combinatorial methods. □

4.3. Higher Layers: Clustered-TSP

Definition 10 (Clustered-TSP Instance)**.** *A TSP instance belongs to a higher layer if the cities can be partitioned into clusters, where intra-cluster distances are small and inter-cluster distances are large. The instance belongs to layer S_m if the cluster structure exhibits hierarchical or recursive properties.*

Clustered-TSP instances can be solved efficiently using hierarchical methods that first solve the TSP within each cluster and then combine the cluster tours into a global tour.

Proposition 9 (Clustered-TSP Construction and Enumeration)**.** *Clustered-TSP instances with a fixed number of clusters can be solved in polynomial time. The number of distinct tour types is related to the number of ways to connect the cluster tours, which can be analyzed using combinatorial methods.*

 doi:10.20944/preprints202509.0961.v1

Proof. The hierarchical structure of clustered instances allows for a divide-and-conquer approach. The TSP is first solved for each cluster, and then the clusters are treated as single super-nodes in a higher-level TSP. This process can be applied recursively for hierarchical cluster structures.

Construction involves defining a set of clusters and assigning cities to clusters with appropriate distance properties. Enumeration of tour types involves classifying the different ways to connect the cluster tours, which can be analyzed using Pólya-theoretic methods for hierarchical structures. □

5. Integer Programming: Kronecker and Unimodular Constructor Layers

Integer Programming (IP) is a general framework for optimization problems with integer variables. General IP is NP-hard [7], but instances with specific matrix structures can be solved efficiently.

5.1. The Separable Layer S_0: Kronecker-IP

Definition 11 (Kronecker-IP Instance)**.** *An IP instance is a Kronecker-IP instance if the constraint matrix A can be expressed as a Kronecker product of smaller matrices: $A = A_1 \otimes A_2$. The objective function and constraint vector must also exhibit compatible separable structure.*

Kronecker-IP instances can be solved efficiently by decomposing the problem into smaller, independent subproblems.

Proposition 10 (Kronecker-IP Construction and Enumeration)**.** *Kronecker-IP instances can be solved in polynomial time by solving the smaller subproblems corresponding to the Kronecker factors. The number of distinct solution types is related to the product of the number of solutions to the subproblems.*

Proof. The Kronecker product structure allows for a change of variables that decouples the original IP into two smaller, independent IPs. These subproblems can be solved separately, and their solutions can be combined to obtain a solution to the original problem.

Construction involves selecting smaller matrices A_1, A_2 and forming the Kronecker product $A = A_1 \otimes A_2$. Enumeration of solution types involves multiplying the number of solution types for the subproblems. □

5.2. The Linear Layer S_1: Unimodular-IP

Definition 12 (Unimodular-IP Instance)**.** *An IP instance is a Unimodular-IP instance if the constraint matrix A is totally unimodular (all square submatrices have determinant 0, 1, or -1). The constraint vector must be integer-valued.*

Unimodular-IP instances can be solved efficiently because the linear programming relaxation has integer optimal solutions.

Proposition 11 (Unimodular-IP Construction and Enumeration)**.** *Unimodular-IP instances can be solved in polynomial time by solving the corresponding linear programming relaxation. The number of distinct solution types is related to the number of vertices of the feasible polytope.*

Proof. The total unimodularity property guarantees that all vertices of the feasible polytope defined by the linear constraints are integer-valued. Therefore, the optimal solution to the linear programming relaxation is automatically an integer solution, and the IP can be solved using standard LP algorithms.

Construction involves creating a totally unimodular matrix, which can be done using various known constructions (e.g., network flow matrices). Enumeration of solution types involves analyzing the vertices of the feasible polytope. □

 doi:10.20944/preprints202509.0961.v1

5.3. Higher Layers: Block-Diagonal-IP

Definition 13 (Block-Diagonal-IP Instance)**.** *An IP instance belongs to a higher layer if the constraint matrix A is block-diagonal with small coupling constraints between blocks. The instance belongs to layer S_m if the block structure exhibits hierarchical or recursive properties.*

Block-Diagonal-IP instances can be solved efficiently using decomposition methods such as Dantzig-Wolfe decomposition or Benders decomposition.

Proposition 12 (Block-Diagonal-IP Construction and Enumeration)**.** *Block-Diagonal-IP instances with a fixed number of blocks can be solved in polynomial time using decomposition methods. The number of distinct solution types is related to the number of ways to combine solutions from the individual blocks.*

Proof. The block-diagonal structure allows the problem to be decomposed into smaller subproblems for each block, with a master problem that coordinates the solutions of the subproblems. This hierarchical approach can be applied recursively for nested block structures.

Construction involves creating a block-diagonal matrix with sparse coupling constraints. Enumeration of solution types involves classifying the different ways to combine solutions from the individual blocks, which can be analyzed using Pólya-theoretic methods. □

6. Graph Coloring: Lattice and Group Constructor Layers

Graph Coloring is the problem of assigning colors to the vertices of a graph such that no two adjacent vertices have the same color. General graph coloring is NP-hard, but instances with specific graph structures can be colored efficiently.

6.1. The Separable Layer S_0: Lattice-Coloring

Definition 14 (Lattice-Coloring Instance)**.** *A graph coloring instance is a Lattice-Coloring instance if the graph is the graph of a regular lattice or grid. The number of colors is related to the properties of the lattice.*

Lattice-Coloring instances can be colored efficiently using simple periodic coloring patterns.

Proposition 13 (Lattice-Coloring Construction and Enumeration)**.** *Lattice-Coloring instances can be colored in polynomial time using periodic coloring patterns. The number of distinct coloring types is related to the number of ways to assign colors to a fundamental domain of the lattice.*

Proof. The regular structure of the lattice allows for a simple coloring pattern that repeats periodically. The number of colors required is determined by the local structure of the lattice (e.g., the degree of the vertices).

Construction involves defining a lattice and its corresponding graph. Enumeration of coloring types involves classifying the different ways to color a fundamental domain of the lattice, which can be analyzed using combinatorial methods. □

6.2. The Linear Layer S_1: Group-Coloring

Definition 15 (Group-Coloring Instance)**.** *A graph coloring instance is a Group-Coloring instance if the graph is a Cayley graph of a finite group G. The number of colors is related to the properties of the group.*

Group-Coloring instances can be colored efficiently using the algebraic properties of the group.

Proposition 14 (Group-Coloring Construction and Enumeration)**.** *Group-Coloring instances can be colored in polynomial time using the group structure. The number of distinct coloring types is related to the number of homomorphisms from the group to the symmetric group on the set of colors.*

 doi:10.20944/preprints202509.0961.v1

Proof. The group structure provides strong constraints on the possible colorings. The coloring problem can be reformulated as a problem of finding a group homomorphism, which can be solved using algebraic methods.

Construction involves selecting a group G and a set of generators to define the Cayley graph. Enumeration of coloring types involves classifying the different group homomorphisms, which can be analyzed using group-theoretic methods. □

6.3. Higher Layers: Modular-Coloring

Definition 16 (Modular-Coloring Instance)**.** *A graph coloring instance belongs to a higher layer if the graph has a modular decomposition into smaller subgraphs. The instance belongs to layer S_m if the modular decomposition is hierarchical or recursive.*

Modular-Coloring instances can be colored efficiently using the modular decomposition of the graph.

Proposition 15 (Modular-Coloring Construction and Enumeration)**.** *Modular-Coloring instances can be colored in polynomial time using the modular decomposition. The number of distinct coloring types is related to the number of ways to combine colorings of the subgraphs.*

Proof. The modular decomposition allows the coloring problem to be broken down into smaller subproblems on the subgraphs. The colorings of the subgraphs can then be combined to obtain a coloring of the original graph.

Construction involves creating a graph with a known modular decomposition. Enumeration of coloring types involves classifying the different ways to combine colorings of the subgraphs, which can be analyzed using Pólya-theoretic methods. □

7. Complexity Event Horizons and Layer Growth Patterns

Our analysis across multiple NP-complete problems reveals consistent patterns in how computational complexity evolves as we move through the constructor layer hierarchy. We introduce the concept of "Complexity Event Horizons" to describe the precise mathematical boundaries where problems transition from polynomial-time constructible structure to exponential-time search requirements.

7.1. The First Horizon: S_0 to S_1 Transition

Empirically, the transition from the separable layer S_0 to the linear layer S_1 represents a fundamental complexity boundary across all problems studied. This transition exhibits several universal characteristics:

Orbit Count Growth: The number of equivalence classes grows from $O(\varphi(k)^2)$ in S_0 to $O(k^2)$ in S_1 for all problems. This quadratic growth pattern appears consistently in Sudoku ($3k^2 - 5k$ classes), SAT (quadratic in variable block size), TSP (quadratic in cluster count), Integer Programming (quadratic in block coupling parameters), and Graph Coloring (quadratic in lattice parameters).

Algebraic Structure Loss: The S_0 layer is characterized by pure multiplicative or separable structure that enables automatic constraint satisfaction. The S_1 layer requires explicit verification of constraints through bi-affine or linear methods, representing a qualitative increase in computational complexity.

Construction Complexity: Construction algorithms transition from $O(k^4)$ time in S_0 to $O(k^6)$ time in S_1, reflecting the additional degrees of freedom and constraint checking required for linear constructions.

Definition 17 (First Complexity Event Horizon)**.** *The First Complexity Event Horizon is the boundary between constructor layers where:*

1. *Orbit counts transition from $O(\varphi(k)^2)$ to $O(k^2)$ growth*

 doi:10.20944/preprints202509.0961.v1

2. *Algebraic structure changes from separable/multiplicative to bi-affine/linear*
3. *Construction complexity increases from $O(k^4)$ to $O(k^6)$*
4. *Constraint satisfaction transitions from automatic to explicit verification*

7.2. Higher Horizons: Piecewise to Search-Based Methods

Beyond the linear layer S_1, higher layers $S_2, S_3, \ldots$ exhibit increasingly complex structure that eventually transitions to search-based methods characteristic of general NP-complete problems.

Piecewise Structure: Layers S_2 and S_3 maintain constructive algorithms through piecewise methods using local pattern catalogs. These layers exhibit super-polynomial growth in the number of equivalence classes but still admit closed-form Pólya enumeration formulas.

Catalog Complexity: The local pattern catalogs $T_{\text{block}}(k)$ that define higher layers grow exponentially in the block size parameter. For sufficiently large k, the catalog enumeration itself becomes computationally intensive, marking the transition toward general search methods.

Assembly Complexity: Global assembly of local patterns becomes increasingly constrained as layer index increases. Eventually, the constraint satisfaction problem for global assembly becomes as hard as the original NP-complete problem, marking the effective boundary of the constructor layer approach.

Definition 18 (Higher Complexity Event Horizons)**.** *Higher Complexity Event Horizons occur when:*

1. *Local pattern catalogs become exponentially large in block parameters*
2. *Global assembly constraints approach the complexity of the original problem*
3. *Construction algorithms transition from polynomial to exponential time*
4. *Pólya enumeration formulas become computationally intractable*

7.3. Universal Growth Patterns

Despite the diversity of problems studied, several universal patterns emerge in layer growth:

Exponential Base Growth: The separable layer S_0 consistently exhibits growth proportional to $\varphi(k)^2$, reflecting the fundamental degrees of freedom in selecting multiplicative parameters from unit groups.

Quadratic Linear Growth: The linear layer S_1 universally exhibits $O(k^2)$ growth, corresponding to the degrees of freedom in bi-affine parameter selection across all problem domains.

Pólya Combinatorial Growth: Higher layers follow Pólya enumeration patterns with growth rates determined by the size of local pattern catalogs and the number of global assembly choices.

Stabilizer Structure: Across all problems, orbit size distributions exhibit the same splitting pattern between large and small orbits, governed by involution structure in parameter spaces.

7.4. Implications for Algorithm Design

The identification of Complexity Event Horizons provides practical guidance for algorithm design:

Structure Recognition: Algorithms should first attempt to recognize whether a given instance falls within the separable layer S_0, where automatic constraint satisfaction enables the most efficient solution methods.

Layer Progression: If separable methods fail, algorithms should systematically progress through higher layers, testing membership in S_1, S_2, etc., until a suitable constructor is found or the instance is determined to lie outside the structured layers.

Hybrid Approaches: For instances that partially match structured patterns, hybrid algorithms can exploit structure where it exists while falling back to search methods for unstructured components.

Preprocessing Benefits: Even when instances do not fall entirely within structured layers, preprocessing to identify structured subcomponents can significantly reduce the effective problem size for subsequent search algorithms.

 doi:10.20944/preprints202509.0961.v1

8. Implications for P versus NP and Complexity Theory

Our framework provides new perspectives on the P versus NP question and the nature of computational complexity more broadly. While we do not resolve P versus NP, we offer insights into the distribution of computational difficulty within NP-complete problems.

8.1. Heterogeneous Complexity Within NP-Complete Problems

Traditional complexity analysis treats NP-complete problems as uniformly hard, with worst-case instances requiring exponential time. Our layer framework reveals that this uniformity is misleading: NP-complete problems contain substantial polynomial-time structure that becomes visible through appropriate mathematical lenses.

The key insight is that computational difficulty is not uniformly distributed within NP-complete problems. Instead, difficulty is concentrated in higher layers of the constructor hierarchy, while lower layers admit efficient algorithms despite the overall NP-completeness of the problems.

Structured Subspaces: Each constructor layer defines a structured subspace of the complete solution space where polynomial-time algorithms are possible. These subspaces are characterized by specific algebraic or geometric properties that enable efficient construction and verification.

Density of Structure: Our computational results suggest that structured subspaces, while representing a tiny fraction of the total solution space, may contain a significant portion of practically relevant instances. This could explain why many real-world NP-complete problems are tractable despite theoretical hardness guarantees.

Complexity Gradients: Rather than a sharp boundary between P and NP, our framework reveals complexity gradients within individual problems. Instances transition smoothly from highly structured (polynomial-time) to moderately structured (quasi-polynomial time) to unstructured (exponential time).

8.2. Implications for P versus NP Resolution

Our results are consistent with both possible resolutions of the P versus NP question:

If P = NP: Our structural hierarchies would collapse, with polynomial-time algorithms existing for all layers. The boundaries we identify would become computational artifacts rather than fundamental barriers. However, the hierarchical structure would still provide valuable insights into algorithm design and problem understanding.

If P $\neq$ NP: Our hierarchies provide a detailed map of the complexity landscape, showing exactly where and why computational barriers arise. The Complexity Event Horizons would represent fundamental limits of efficient computation, with higher layers requiring exponential time in the worst case.

Practical Implications: Regardless of the resolution of P versus NP, our framework provides actionable insights for algorithm design and problem-solving strategies. The identification of polynomial-time structure within NP-complete problems has immediate practical value.

8.3. Connections to Parameterized Complexity

Our layer framework exhibits strong connections to parameterized complexity theory, which studies how problem difficulty depends on specific parameters:

Fixed-Parameter Tractability: Many of our constructor layers correspond to fixed-parameter tractable cases where specific structural parameters are bounded. For example, separable Sudoku instances are parameterized by the choice of multiplicative generators (α, β).

Kernelization: The preprocessing algorithms that recognize layer membership can be viewed as kernelization procedures that reduce instances to their essential structural components.

Parameter Hierarchies: Our layer hierarchy provides a natural parameterization where higher layers correspond to more complex parameter choices, eventually transitioning to the general (unparameterized) NP-complete problem.

 doi:10.20944/preprints202509.0961.v1

8.4. Algorithmic Meta-Principles

Our analysis suggests several meta-principles for designing algorithms for NP-complete problems:

Structure-First Approach: Algorithms should prioritize recognition and exploitation of structural properties before resorting to general search methods. The potential for polynomial-time solutions within structured layers justifies this approach.

Hierarchical Decomposition: Complex instances should be decomposed hierarchically, with structured components handled by specialized algorithms and unstructured components addressed through search.

Adaptive Complexity: Algorithm complexity should adapt to instance structure, using efficient methods for structured components and more expensive methods only when necessary.

Structural Preprocessing: Even when instances do not fall entirely within structured layers, identifying and exploiting partial structure can significantly improve algorithm performance.

8.5. Broader Implications for Complexity Theory

Our framework suggests several directions for advancing complexity theory:

Fine-Grained Complexity: Rather than coarse-grained classifications like P and NP, we need fine-grained analyses that capture the spectrum of computational difficulty within individual problems.

Structural Complexity Classes: New complexity classes could be defined based on the types of structure that enable efficient algorithms, providing a more nuanced view of computational difficulty.

Average-Case Analysis: Our results suggest that average-case complexity may be significantly better than worst-case complexity for many NP-complete problems, due to the concentration of structure in practically relevant instances.

Approximation and Heuristics: Understanding the distribution of structure within NP-complete problems could lead to better approximation algorithms and heuristics that exploit common structural patterns.

9. Conclusion and Future Directions

Open problems.

- **Layer completeness.** Prove or refute that every feasible instance lies in some finite constructor layer S_m with finite local catalogs; identify minimal m for broad classes.
- **Catalog growth.** Determine tight asymptotics (in k) for the size of catalogs at S_1 and higher layers; characterize when growth is polynomial vs. superpolynomial.
- **Orbit-width scaling.** Classify how orbit-width and spectral sparsity scale on random or adversarial SAT/TSP instances; relate thresholds to tractability of membership/enumeration.

We have developed a unified framework for analyzing polynomial-time structure within NP-complete problems through constructor layer hierarchies. Our approach reveals that computational difficulty is not uniformly distributed within NP-complete problems, but rather concentrated in higher layers while substantial polynomial-time structure exists in lower layers.

9.1. Summary of Contributions

Our main contributions include:

Theoretical Framework: We established a rigorous mathematical framework for analyzing constructor layers within NP-complete problems, providing tools for systematic enumeration and classification of structured solution families.

Sudoku Analysis: We provided comprehensive theoretical and computational results for Sudoku-like constraint satisfaction problems, including closed-form formulas for separable and linear solution classes validated through extensive computation up to $k = 20$.

Cross-Problem Generality: We demonstrated the generality of our approach by extending the framework to Boolean Satisfiability, Traveling Salesman Problem, Integer Programming, and Graph Coloring, showing consistent patterns across diverse problem domains.

 doi:10.20944/preprints202509.0961.v1

Complexity Event Horizons: We introduced the concept of Complexity Event Horizons to describe precise mathematical boundaries where problems transition from polynomial-time constructible structure to exponential-time search requirements.

Algorithmic Insights: We provided polynomial-time construction algorithms for structured layers and demonstrated how structure recognition can guide algorithm design for NP-complete problems.

9.2. Open Problems and Future Research

Several important questions remain open for future investigation:

Layer Completeness: Prove or disprove that the constructor layer decomposition $S_0 \subset S_1 \subset S_2 \subset \cdots$ eventually covers the complete solution space Ω for each NP-complete problem studied. This fundamental question determines whether our framework provides a complete structural classification.

Asymptotic Growth Rates: Establish tight asymptotic bounds for the growth of layer catalog sizes as problem parameters increase. While we have empirical evidence for quadratic growth in S_1 layers, rigorous theoretical analysis of higher layers remains open.

Orbit-Width Classification: Develop a complete classification of orbit-width parameters for random instances of SAT, TSP, and other NP-complete problems. This would provide theoretical foundations for the conditional tractability program outlined in Section 8.

Structural Recognition Algorithms: Design efficient algorithms for recognizing when a given instance belongs to a specific constructor layer. While we provide construction algorithms, the inverse problem of structural recognition requires further development.

Cross-Problem Universality: Investigate whether the structural patterns observed across our five example problems (Sudoku, SAT, TSP, IP, Graph Coloring) represent universal features of NP-complete problems or are specific to constraint satisfaction and optimization domains.

Theoretical Foundations: Can we prove general theorems about the existence and properties of constructor layer hierarchies for arbitrary NP-complete problems? What mathematical conditions guarantee the existence of polynomial-time structured layers?

Complete Characterization: For Sudoku and other specific problems, can we provide complete characterizations of all constructor layers, including precise formulas for higher layers beyond S_2 and S_3?

Algorithmic Applications: How can our structural insights be translated into practical algorithms for solving real-world NP-complete problems? Can we develop automated tools for identifying and exploiting structure in arbitrary problem instances?

Funding: No external funding has been received.

Data Availability Statement: All results are theoretical and no additional data has been produced.

Acknowledgments: The author acknowledges the use of AI assistance in refining the mathematical formulations and computational validations presented in this work. All theoretical results, proofs, and interpretations remain the responsibility of the author.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.
2. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
3. T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E86-A(5):1052–1060, 2003.
4. C. J. Colbourn. The complexity of completing partial Latin squares. *Discrete Applied Mathematics*, 8(1):25–30, 1984.

 doi:10.20944/preprints202509.0961.v1

5. G. Pólya. Kombinatorische anzahlbestimmungen für gruppen, graphen und chemische verbindungen. *Acta Mathematica*, 68(1):145–254, 1937.
6. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
7. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.